Discriminability of simple and complex haptic vibrations in single-cell computational and human psychophysical settings

A THESIS
SUBMITTED TO THE FACULTY OF
UNIVERSITY OF MINNESOTA
BY

Nicholas D. Theis

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Victor H. Barocas

July 2017

Acknowledgements

Portions of this work were conducted using computational resources provided by the Minnesota Supercomputing Institute. This project was funded in part by the NSF Intergrative Graduate Education and Research Traineeship (IGERT). I wish to thank Abbigal Vandusen for establishing the preliminary apparatus and protocols used in these psychophysical experiments, and Brendan Young-Dixon for developing code. I also wish to thank Louis Ting and Tiffany Senkow for their technical contributions, as well as all the volunteers who participated in this study. A special thanks to Julia C. Quindlen (JCQ), the IGERT recipeint, for developing the computaional model, and for offering a breadth of advise and technicial support throughout, including scripts used to operate the DAC apparatus from MATLAB. Last, and not least, I want to thank Dr. Victor Barocas (VHB), whose inviting mentorship made this project possible. An extended thanks to JCQ and VHB for providing numerous revisions and feedback on the technical and stylistic content of this writing.

Dedication

This thesis is dedicated to former Minneapolis mayoral candidate and University of Minnesota Twin Cities campus icon, Mike Gould.

Abstract

A multiscale, multiphysics model of the Pacinian Corpuscle (PC) was used to study the neurophysiological response to haptic vibrations in the 100-200Hz range. The computational results were compared to human psychophysical experiments, emulating the pairing of psychophysics with in vivo electrophysiology in PC research. A first assessment of this approach was made by examining the discriminability (d') of pairs of vibrotactile stimuli. The discrimination task was performed psychophysically and in silico for both one- and two-frequency stimuli. Both firing rate and inter-spike interval neural decoding schemes were used to calculate d' from simulation data. Human subjects discriminated between frequencies with two components (complex stimuli) more effectively than isolated frequencies (simple stimuli), possibly due to the presence of beat frequencies in dissonant stimuli. Over a given stimulus set, in silico d' values correlated well with the psychophysical data ($R2 > 0.6$), but when the simple and complex data were combined the model did not match the experiment ($R2 < 0.1$). Firing rate resulted in better predictions than inter-spike interval, and was more robust to noise. Results suggest that a single simulated PC can capture some but not all of the observed psychophysical response to a vibrotactile stimulus.

Table of Contents

List of Figures

1 Introduction

1.1 Study Background Motivation

The Pacinian corpuscle (PC) is a dermal touch receptor responsible for transducing high-frequency vibrations, making it a central biological component for haptic processing. PC afferents primarily innervate the palm and fingers, but they can also be found in other areas of the body [1]. PCs have limited spatial resolution, but are sensitive to skin deformations as small as 10nm [2]. Discovered nearly two centuries ago [3], the PC along with related touch receptors have seen a recent resurgence of interest driven in part by the pursuit of better haptic technology and from an effort in neuromodulation research towards developing somatosensory prosthetics [4] [5] [6] [7]. A revised understanding of movement disorders as part sensory disorder [8] offer further biomedical applications of haptic technology in relation to PC physiology, for instance in the form of at-home patient monitoring (telemedicine) or therapy regimes.

Previous studies of the neural representation of touch stimuli have often relied on behavioral experiments combined with electrophysiological recordings [9] [10]. Early experiments studied pressure-membrane-voltage relationships (for instance, response thresholds and tuning thresholds), often comparing human psychophysics (e.g. perceptual thresholds) to the nervous response in an anesthetized animal [10] [11] [12]. Later work examined the PC's ability to

encode more complex vibrations, in particular polyharmonic stimuli, and made efforts to describe results with functional models and to make inferences about the neural code [9] [13]. Following this general experimental paradigm, the goal of the present study was two-part: (1) to implement a recently developed mechanistic model of a single PC [14] in place of traditional *in vivo* electrophysiology, and (2) to assess this approach by comparing relevant psychophysical measures of various stimuli to model predictions of the same stimuli.

The model used in this study is a multiphysics computer simulation of the PC, which captures both biomechanical and electrophysiological properties of the receptor [14]. The model involves several scales of the receptor transduction process (**Fig. 1**). In the adult human hand, each PC afferent ends in a single ellipsoidal corpuscle, usually about 3-4mm in length [15]. The outer core of the corpuscle consists of several lamellar layers (approximately 30 in healthy adults) of collagen-associated [16] epithelial-like cells spaced by fluid [17]. This structure is modeled as spherical shells interspersed with fluid, using equations form shell theory and lubrication theory [14]. Biologically, and in the computational model, these lamellae act as a high-pass filter, removing low frequency vibrations that would otherwise overwhelm the nerve.

At the center of the outer core, a densely packed inner core of lamellae derived from Schwann cells surrounds the PC neurite [18, 19]. The inner core and neurite are modeled as a solid in the COMSOL environment in the simulation

[14]. The surface of the neurite contains membrane protrusions (filopodia) hypothesized to be the sites of mechanically-gated ion channels [20]. The stretching of these ion channels leads to action potentials in the nerve afferent; this stage in the signal transduction process is modeled in the NEURON simulation environment.

A detailed understanding of the relationship between PC structure and function via a modeling approach has motivated several studies over the past fifty years [21] [22] [23]. A sufficiently detailed model, for example, could provide insight into the meaning of structural changes observed during normal development and aging, or in pathological conditions. Perhaps the most notable disease state involving the PC is Dupuytren's contracture, a condition where the fingers are continuously pulled into a bent position due to the presence of tight cords under the skin. These cords develop from knots over a period of years and coincide with altered PC morphology [24] [25]. The present study was motivated in part by such considerations. Specifically, this study seeks to develop a methodology by which a computational model can be related directly to human subjects. It is hypothesized that a sufficiently detailed model can capture behavioral, in particular psychophysical, responses of human subjects to stimuli that excite the modeled system. While this is neither a development nor disease-state study, the methodology developed here offers an outline of how such future research could be conducted using a paired psychophysical-computational paradigm.

1.2 Pacinian Corpuscle Background

For haptic sensations, like all sensory modalities, specialized cell types transduce physical phenomena into electrophysiological signals. In the case of vibrotactile stimuli, these specialized cells are either of the “rapid adapting” mechanoreceptor class (RA; sometimes “fasting adapting,” FA) consisting of the Meissner’s corpuscle (MC) which is sensitive to low frequency vibrations (i.e. 10-50 Hz), or the Pacinian corpuscle class (PC; also known as Lamellar corpuscle) which is activated at smaller indentations, and higher frequencies (30 – 1000Hz), being most sensitive at 250 Hz [2] [10] [11]. MCs and PCs play different functional roles in touch, MCs measuring movements over the whole hand, like slippage, and PCs provide a sensation of distant activity, like the presence of dirt moving on the end of a shovel or the action at the end of a carving knife.

PCs have almost no spatial resolution but are highly sensitive to skin deformations (as small as 10nm, as noted above) [2]. This minimum indentation amplitude is known as the response threshold, that is, the smallest indentation amplitude that can elicit an action potential in the nerve. At a second, and higher threshold, namely the tuning threshold, the PC has a strong tendency to phase-lock its spike rate with single-component stimulus frequencies [2]**.** The model used in this study has previously been shown to exhibit these threshold-response features [14].

From a computational neuroscience perspective, the tuning threshold property is suggestive of a rate code, where firing rate indicates the frequency of

the stimulus and overall recruitment in the population captures stimulus intensity. Not all natural stimuli of course are a single-frequency pure sinusoid, but more often noisy, multi-component vibrations, which suggests in principal that PCs must rely on a more elaborate code if they are to interpret a wide gamut of natural sensations. Earlier work has concluded that stimulus information is derived from the mean inter-spike interval (ISI) of afferent cells, a finding that is not necessarily at odds with the rate code hypothesis, since ISI is essentially instantaneous firing rate [9]. It is possible, and indeed likely, that neither of these crude coding schemes completely captures the manner in which the nervous system processes spiking activity, which duly involves population coding. ISI and FR are nonetheless two exemplary coding schemes for processing the (simulated) biological output, and so will be used here as such. Some similarities and differences of these coding schemes with respect to this experimental design will be examined throughout this document.

Earlier studies of vibrotaction have been motivated in part by insights into haptic processing that are offered via an analogy to hearing. The cellular and molecular mechanisms that encode these external stimuli are remarkably similar, and this commonality holds true across the animal kingdom [26] [27]. Hair cells in the ear as well as many touch receptors types in the skin rely on mechanically gated ion channels to transduce deformations and vibrations into membrane currents and ultimately action potentials for further processing by the central nervous system. At the neural network level, it has been suggested that cuneate

nucleus neurons, where touch receptors project, extract touch feature information (such as curvature) using a mechanism similar to inter-aural time difference for sound localization [28].

However, despite the shared biological mechanisms and similar stimulus energy state this analogy is incomplete. The somatotopy (spatial mapping) of the touch system in contrast to the tonotopy (frequency mapping) of the auditory system is a crucial incongruity, resulting in the much better spectral discrimination of the hearing sense. This consideration has motivated some studies of the response of the PC to complex vibrations, including this study, in part. Despite the worse spectral discrimination of touch, haptic sensations can nonetheless be used to restore some music perceptions in individuals with hearing-impairments, although with a narrower range of notes, and smaller variety of waveforms [4]. These higher order, and multimodal aspects of sensation and perception are not the focus of this work. A more complete understanding of PC biology, even on the individual receptor level, will nevertheless lend insight into the application of haptic feedback for somatosensory prosthesis applications, the rehabilitation of other senses through haptic feedback, and possibly the design of biomimetic algorithms [7].

1.3 Clarification of Stimulus Notation Used

In both the *in silico* and psychophysical experiments, stimuli were drawn from two stimulus sets: *simple* and *complex*. *Simple* stimuli consisted of single-

frequency pure sinusoid waves in the 110-190Hz range. *Complex* stimuli were formed by adding an equal amplitude 100Hz component to each waveform from the simple set. This 110-190Hz range represents the lower end of the PC sensitivity range, where it has been suggested that a single PC is unable to encode complex stimuli [2].

Hereafter, a stimulus is named by its frequency, followed by a “C” for complex stimuli or an “S” for simple stimuli. For example, 140C refers to a complex stimulus formed by combining a 100Hz wave with a 140Hz wave (both sinusoidal). 140S refers to a pure 140Hz sinusoid. The simple stimulus set consisted of 110S, 140S, 150S, 160S, and 190S; the complex stimulus set consisted of 110C, 140C, 150C, 160C, and 190C (**Fig. 2**). Stimuli were only compared within sets; simple stimuli were not compared directly to their complex counterparts. Notably, complex waveforms of two components undergo phases of constructive and destructive interference, resulting in an enveloped waveform, where the frequency of constructive phases, or beat frequency, is equal to magnitude of the difference between the two components.

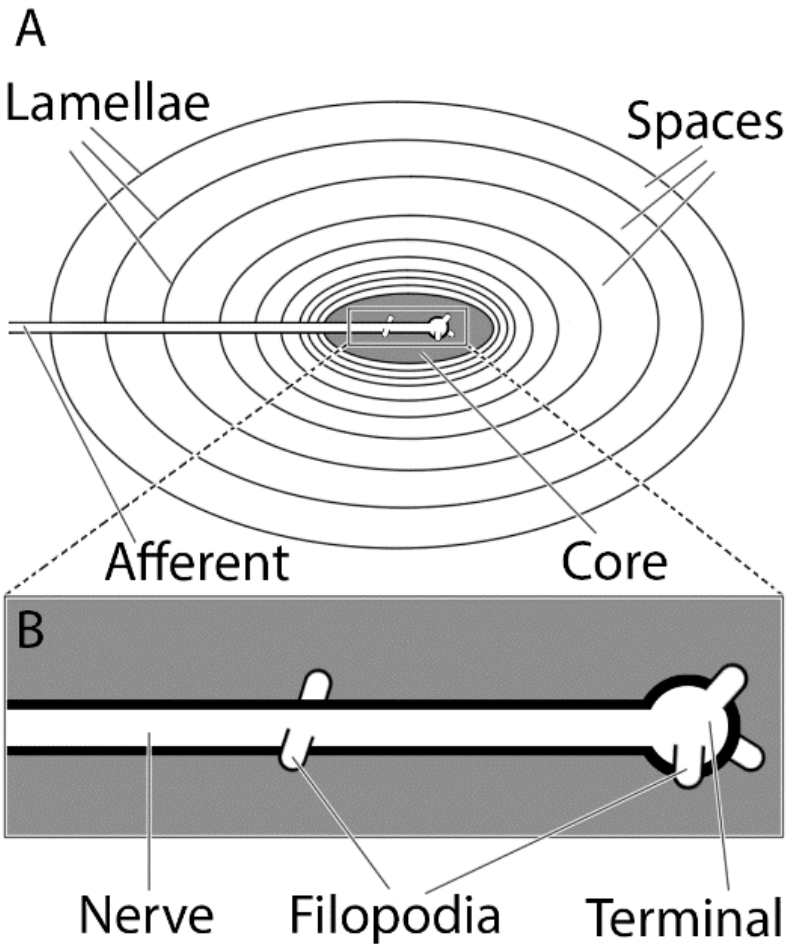

Figure 1. Simplified diagram of the PC. (A) Entire corpuscle. The concentric lamellae that make up the outer corpuscle layers, interspersed with fluid-filled spaces, act as a high-pass filter of mechanical vibrations. Forces are transmitted to the densely-packed inner core, shown in grey. The nerve afferent exiting the corpuscle is a myelinated axon. **(B) Zoomed-in view of the nerve ending.** The neurite within the core inner core region is considered bare (not myelinated). The neurite's geometry is modeled as a bulbous nerve terminal and five filopodia (two proximal, three distal) assuming different orientations.

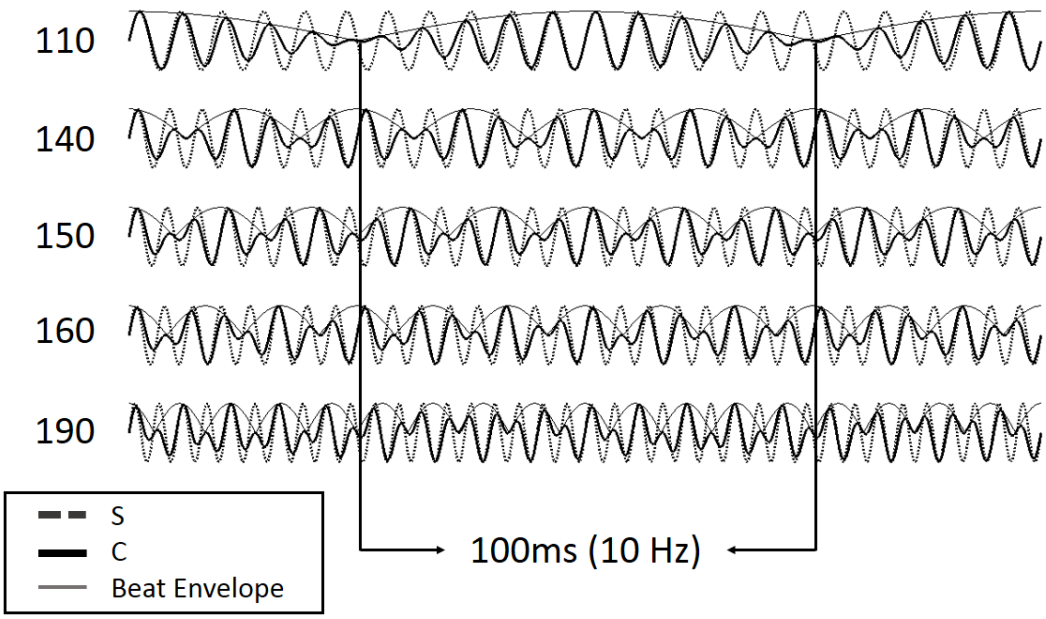

Figure 2. Stimuli used in psychophysical and *in silico* experiments. The 10 stimuli tested, from two stimulus sets, are shown for a 200ms window. Light dotted plot lines represent simple (S) waveforms, heavy solid lines represent complex (C) waveforms, and light solid lines represent the upper envelope of the beat (interference pattern). The 100ms (10Hz) region corresponds to 1 beat of the 110C waveform. Note that this region also corresponds to 4 periods of the 140C beat, 5 of the 150C beat, and so on.

2 Psychophysical Experiments

2.1 Psychophysical Methods

The psychophysical experiments were designed so that a measure of perceived difference between a pair of haptic stimuli could be gleaned from a small population of human subjects. This is achieved through methods developed in the field of detection theory [29] and other established behavioral protocols from psychophysics research, as detailed below. Experiments were Institutional Review Board (IRB) approved and carried out under IRB guidelines.

Nine adult, right-hand-dominant subjects (four male, five female, ages 18-37) participated in psychophysical testing. Subjects were seated and asked to place their right index finger on a piezoelectric actuator (Adafruit, New York) without applying pressure. Vibrotactile stimuli were then delivered to the subjects by the actuator via a digital-to-analogue signal generator (Syscomp WGM-201, Ontario), which was controlled using custom MATLAB scripts. Analogue signals to the actuator were amplified (Gemini XGA-3000, New Jersey) and amplifier volume was not altered between experiments or subjects. Subjects were prompted by MATLAB to answer questions about their perceptions of the stimuli. During all testing procedures, subjects wore headphones playing pink noise to mask the sound of the actuator and other ambient auditory sounds that might distract them from their decision making, or provide them with unintended auditory cues. MATLAB codes used to run experiments are provided in the appendices section, as described below.

2.1.1 Apparatus

The digital to analogue signal generator (DAC), a Syscomp WGM-201, was controlled using MATLAB by a USB interface. MATLAB code, written by JCQ, which was used to operate the DAC is available in **Appendix A.** The Syscomp WGM-201 can take inputs for custom waveforms, and this feature was used in the experiments detailed below to deliver complex stimuli. The DAC requires custom inputs to be specified as .dat files containing 256 8-bit bytes [30], i.e. 256 time samples with an integer value between 0 and 255. Consequently, custom waveforms, namely any *complex* waveform (in the sense used throughout this document), need to be quantized, and this digitization process could create aliasing issues for some frequency combinations.

Consider the stimulus consisting of a 100Hz component and a 110Hz component. Both frequencies share a 10Hz fundemental frequency (both are integer multiples of 10). One period of a 100Hz wave corresponds to 1.1 periods of a 110Hz wave, and 10 periods of 100Hz correspond to 11 periods of a 110Hz, and so on. This is pictures in **Fig. 3A.** Since both frequencies share a 10Hz fundemental, the waveform that represents a 100Hz+110HZ stimulus is the same waveform that represents 10Hz+11Hz stimulus, the former is merely played 10 times faster. Therefor, to create the 110C stimulus, 10 periods of a 10Hz sinusoid are added to 11 periods of a 11 Hz sinusoid, and the resulting waveform is played 10 times per second (i.e., at 10Hz) to achieve a stimulus (2-component) frequency of 100Hz+110Hz.

For frequencies that are not both of base 10 (**Fig. 3B**), a greater number of cycles is needed before the two frequencies will synch at both the first and last sample, that is, they will synch to produce a periodic waveform. More specifically, 10 cycles of a 100Hz wave correspond to 13.6 cycles of a 136 Hz wave. Thus, to achieve a completely faithful reconstruction of this more naunced 2-component waveform, the 100Hz component will need to be carried out for 100 cylces, and the 136Hz component carried out to 136 cycles. However, this waveform needs to be sampled to 256 points in order to be inputed into the DAC as a custom waveform. One of the frequency components, which is represented by 100 cycles of a sinisoid, can be sampled at this rate. But the Nyquist frequency of the 136Hz signal is 272 samples/second, and we are only given 256 samples.

These considerations show that our choice of stimulus is somewhat limimted. It is possible alternative approaches could avoid these limitation entirely, such adding two anaolgue signals instead of creating complex waveforms digitally. But for these purposes, it is sufficient to only compare the two-frequency complex waveforms that are composed of base-ten frequency components, since even with this restraint there are still too many possible stimuli in the lower sensitivity range of the PC than can be reasonbly tested. Nonetheless, the limitations of the apparatus just detailed partially informed the choice of stimuli that are examined throught this document.

2.1.2 Thresholding

All subject's thresholds were determined for simple stimuli using an up-down procedure [31]. Briefly, subjects were presented with a stimulus at a higher amplitude than their suspected threshold (in this case, 25% maximum amplitude of the controller output). Subjects were asked whether or not it was perceptible, and if so, the stimulus amplitude was decreased by a step-size (equal to 5% of the maximum controller output) until the user reported it was imperceptible, at which point the amplitude was step-wise increased until the user could again detect the stimulus. This process was repeated for a total of two up-down cycles. After the first cycle the step size was halved to provide a more accurate measure of the true threshold. Original MATLAB code for automating this procedure is presented in **Appendix B.**

All stimuli in subsequent experiments were delivered at an amplitude of four times the highest threshold for a given subject (the frequency they were least sensitive to, usually 100Hz). All complex stimuli were delivered such that the maximum signal amplitude of the complex stimulus was also equal to this four-times max threshold value for simple stimuli. These stimuli were assumed to only excite PCs since they all lie within the PC sensitivity range. Furthermore, the stimuli were presented at only four times PC thresholds, so even beat-frequencies, which may be as low as 10Hz, are assumed to only affect PC

afferents, since RA afferents (which are receptive to these lower frequency vibrations) are about one hundred times less sensitive than PCs [2].

Thresholding was conducted to screen out individuals with abnormally high thresholds (none were found), train subjects on the experimental set-up, and to normalized small difference in individual sensitivity curves. Thresholding was not conducted to establish the precise psychophysical relationship between indentation amplitude and perception. Some statistical significance between subject's thresholds were found but these individual differences are normalized in the discriminability experiments.

2.1.3 Same-Different Experiments

A Same-Different test, as described by Macmillan and Creelman [29] was used to assess the discriminability between stimuli. Subjects were presented with two short-duration (1 second) stimuli in short succession (separated by a 0.5-second silent period). The stimuli were either the same or different; the order in which pairs were presented was randomized. After being presented with a stimulus pair the subject must respond "same" or "different" based on their perception of the sensations before moving onto the next comparison. Each stimulus set of 5 vibrations (110S, 140S, 150S, 160S, 190S, or 110C, 140C, 150C, 160C, 190C) provided 10 two-way comparisons, summarized by 10 d' values per stimulus set. MATLAB scripts used to conduct Same-Difference Experiments are presented in **Appendix C.**

The d' value (**Eq 1**) for two stimuli, S1 and S2, is given as a function of the probability of a correct response, p(c), where z is the inverse normal cumulative distribution function.

$$d' = 2z(\tfrac{1}{2}(1 + \sqrt{2p(c) - 1})) \qquad (1)$$

In this equation, if p(c) was less than 0.5 (which represents a percent correct that is no better than chance), d' was set to be equal to zero. The probability of a correct response, p(c), is determined from the hit rate (H) and false rate (F) for a given stimulus pair (**Eq 2**).

$$p(c) = p(different)H + p(same)(1 - F) \qquad (2)$$

Here, H is the probability that the subject responds “different” when the stimuli are indeed different, and F is the probability the subject responds “different” when the stimuli are in actuality the same. In this experiment, the probability that the stimulus pair was different, p(different), and the probability that the stimulus pair was the same, p(same) were both 0.5, since observations were evenly split between the same and different conditions.

The total observations used to determine H and F for each subject were both 4, so subject-wise d’ values were based on 8 observations. In this study, H and F responses were pooled from all 9 threshold-normalized subjects, therefore each d' value determined using psychophysical testing was based on 72 Same-Different observations across 9 subjects. The MATLAB scripts that were used to pool experimentally determined H and F values, and to determine d' from H and F can be found in **Appendix D.**

2.2 Psychophysical Results

Averaged threshold values for all subjects are shown in **Fig. 4A**. While threshold values at higher frequencies tended to be lower, the effect was small compared to individual variability. Results of the same-different discrimination experiments are shown in **Fig. 4B** and **Fig. 4C**. As can be seen in **Fig. 4C**, the simple stimulus frequencies that were further apart (further from the diagonal of the chart) tended to have higher discriminabilities, however this relationship was dampened at lower frequencies: 140S and 150S had a d' of zero, which represents chance discriminability, but 150S and 160S had a non-zero d'. 110S and 140S were the same distance apart as 160S and 190S (a 30Hz difference in both cases), but the lower frequency pair was not discriminable and the higher pair was highly discriminable.

Complex stimuli were generally more discriminable when their beat frequencies were farther apart (**Fig. 4B**). Stimuli involving a comparison with 110C, however, were more highly discriminable than comparisons not involving 110C with the same difference in beat frequencies, for instance the d' of 110C and 150C (d' = 2.66) versus 150C and 190C (d' = 1.44). This may be because a base frequency of 100Hz was used as the second frequency component in all complex cases, which resulted in the 110C stimulus being highly dissonant (having a slow beat frequency). Discriminabilities of complex stimuli were in general higher than those of their simple counterparts (**Fig. 4D**). This finding was

significant in a 1-tailed paired t-test ($p < 0.01$). Only in one case, 160C versus 190C, was the single frequency more discriminable than the complex frequency.

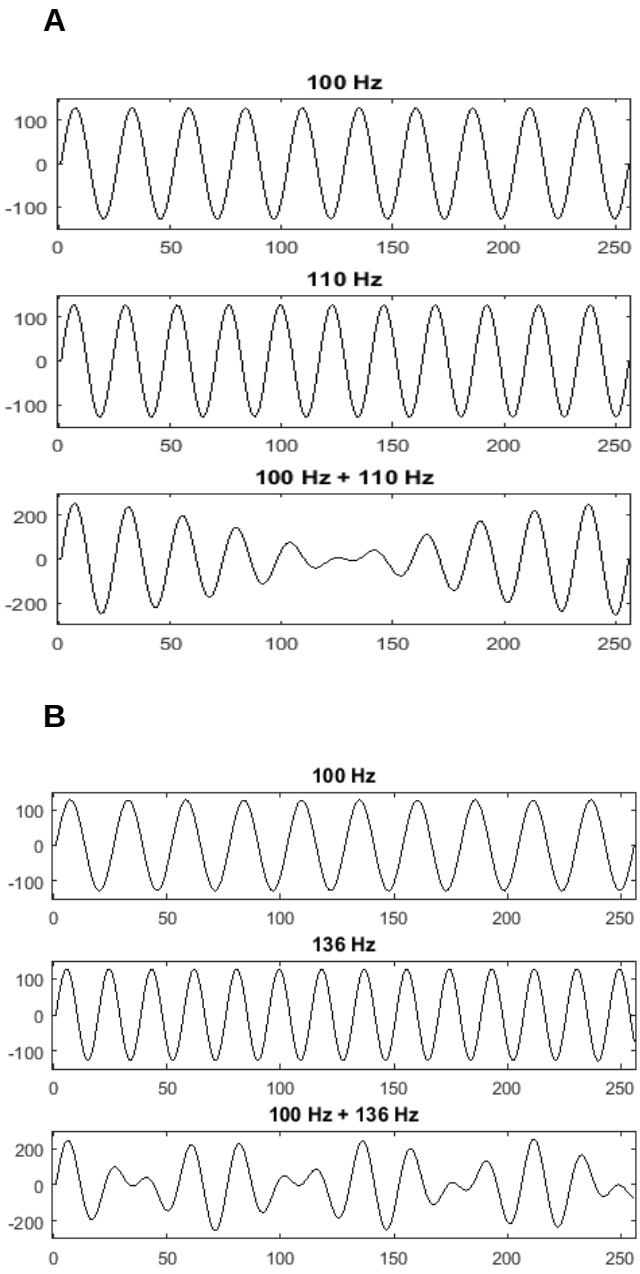

Figure 3. Constructing digital complex waveforms. A. 100ms of a 100Hz and a 110Hz waveform, which share a 10Hz fundamental frequency, are added to produce one period of an 110C waveform. **B.** 100ms of a 100Hz and a

136Hz waveform, which do not share the 10Hz fundamental, are added, producing a non-periodic, faulty 136C waveform.

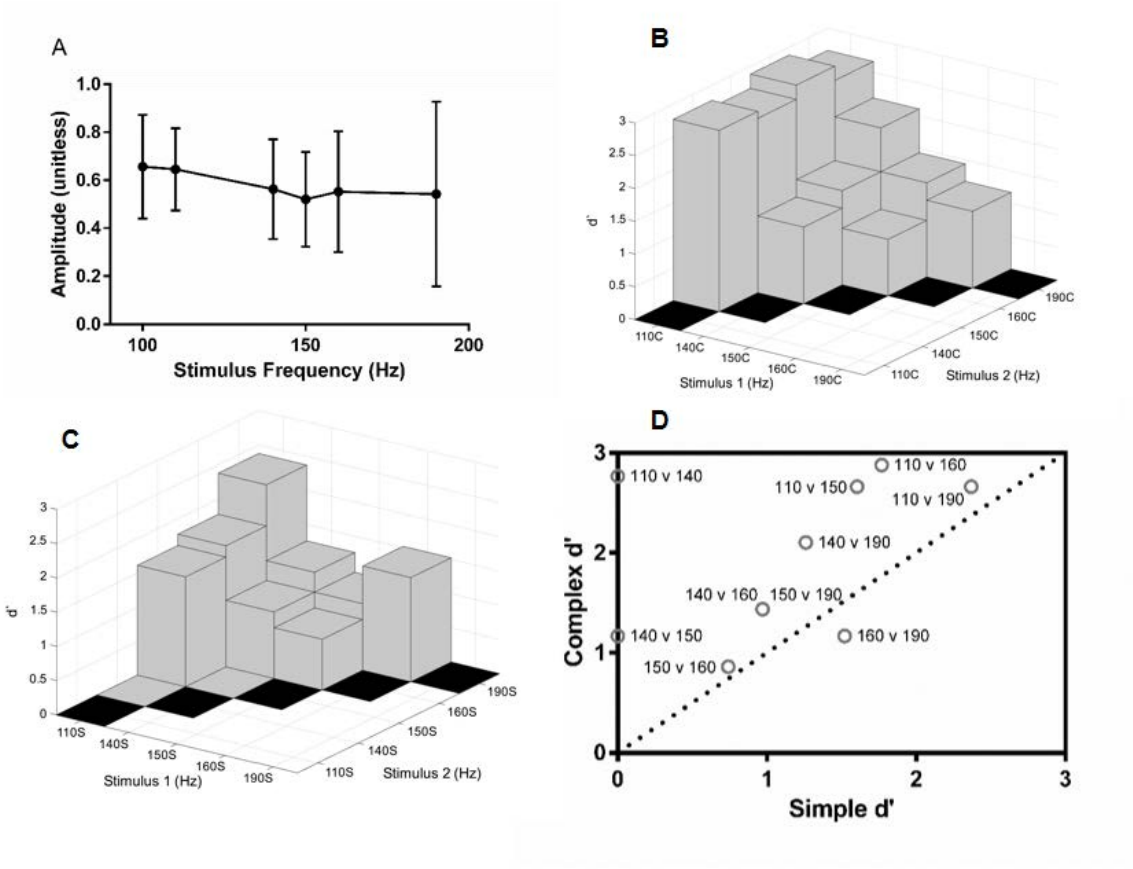

Figure 4. Psychophysical results. (A) Mean threshold ± standard deviation for 9 subjects at the 6 frequencies tested (100S, 110S, 140S, 150S, 160S, and 190S). Amplitude is scaled to the maximum output for the system. **(B)** Discriminability (d ') for each frequency comparison in the complex (with 100Hz base frequency) and **(C)** simple (single frequency) cases. Black tiles in both **(B)** and **(C)** indicate the diagonal; the discriminability of a stimulus from itself is trivially zero. **(D)** Discriminability of simple vs. complex stimuli for each discriminability pair. Pairs are labeled on the plot, for instance 140 v 190, indicating the two frequencies being compared, where the horizontal coordinate is the d' of 140S and 190S (d' = 1.26), and vertical coordinate is the d' of 140C

and 190C (d' = 2.11). Two pairs, 140 v 160 and 150 v 190 occupy nearly the same point in this space. The dotted line represents the line x=y. Points along this line are equally discriminable in the simple and complex case; points above are more discriminable with the 100Hz base (i.e. as complex stimuli), whereas points below are less discriminable in the complex case.

3 *In Silico* Experiments

3.1 Simulation Methods

Calculating d' using H and F, as described in the previous chapter, provides an estimate of d', i.e. the distance, scaled by their standard deviations, between the means of two normal distributions, as in **Eq. 3**, where μ_1 and μ_2 and σ_1 and σ_2 are the means and the standard deviations of the encoded response to S1 and S2, respectively.

$$d' = \frac{\mu 1 - \mu 2}{\sqrt{\frac{1}{2} * (\sigma 1^2 + \sigma 2^2)}} \qquad (3)$$

Clearly these statistics are not always available, necessitating an estimate of d' via H and F in the psychophysical setting. For an excitable membrane however, these values can be determined from the distributions of action potentials over time. *In silico* mean and standard deviations can be calculated based on the distributions of simulated outputs to a given stimulus. In this study, d' values were calculated based on the simulated neuronal firing rate per 100ms (FR_{100}) as well as the inter-spike intervals (ISIs). This simulated neural decoding process is described in the following sections, as well as further explanation on how to operate the model.

3.1.1 Current Injections

Simulated mechanical stimuli were applied to the outermost lamellae layer (shell), and propagated to subsequent (deeper) shells through the fluid-filled spaces between lamellae. This process was modeled in MATLAB, as previously described [14]. The deflection of the inner-most shell was then applied to the inner core of the PC, modeled in COMSOL as an incompressible solid sphere surrounding an isotropic linearly elastic neurite with a complex geometry. Once mechanical stimuli reached filopodia, where mechanically gated ion channels are located in the model, the stimuli were transduced into electrical (nervous) signals, i.e. action potentials. Membrane currents were calculated from filopodia strains normal to the membrane surface using a sigmoidal relationship as previously described [14]. Strain waves were rectified by zeroing negative values. Each of the five filopodia had a different orientation. For MATLAB code used to add filopodia strains, see **Appendix E**. For sigmoid processing code see **Appendix F**.

Current injects were scaled so that simple stimuli resulted in a spiking rate equal to stimulus frequency (the tuning threshold). This allowed for the modeled stimuli, like the stimuli in the psychophysical setting, to be delivered at a volume relative to a measureable threshold. Complex stimuli were produced by adding two single-frequency strain components, which were rectified after they were added. Each component of the complex wave was scaled to half the tuning threshold value, so that their summed amplitude during constructive interference would be equal to the amplitude of simple waves. These operations are

allowable because the biomechanical stages of the model are linear (unlike the electrophysiological stages).

3.1.2 Spike Counting

The simulation environment NEURON was then used to generate voltage traces at the axon from membrane current injections at five dendritic locations, representing filopodia, where mechanically-gated ion channels are located in the model and where strains are the highest on the neurite, as previously described [14]. MATLAB was used to create neuron files from a skeleton .hoc script, which is given in **Appendix G**. Simulated voltage traces were processed in MATLAB using its `findpeaks` function, defining action potentials as peaks occurring above 0 mV. For each simulation, all spike times were determined for the entire 1000ms duration of the simulation. These peak locations were then used to calculate ISI and FR_{100} historgrams. The means and standard deviations of these ISI and FR_{100} observations for 10 repetitions per stimulus were then calculated, and the stimuli were compared using **Eq. 3**. MATLAB scripts for this procedure are available in **Appendix H**.

3.1.3 Addition of Noise

Noise was added to the strains on each individual filopodia prior to converting these strains to currents and passing the currents through the NEURON model. Noise was modeled as a normal distribution with mean of zero

and standard deviation equal to a “noise level” parameter (ranging from 0 to 1) which was varied to produce different SNR values. This was implemented using the MATLAB `normrnd` function. Ten repetitions were performed for each SNR value for every stimulus. SNRs reported represent the average of the five filopodia.

3.2 Simulation Results

PC activity was simulated for the five simple and five complex stimuli examined in the psychophysical setting. **Figure 5** depicts one example complex stimulus waveform (**Fig. 5A**) and the simulated nervous response (**Fig. 5B**). The simulated neuron was observed to closely follow the stimulus waveform by locking its firing rate to the stimulus. For instance, simple stimuli were always found to elicited a spiking rate equal to stimulus frequency when stimulated at high enough amplitude (specifically, at or above the tuning threshold), a long-known characteristic of the PC.

Due to their individual orientations, some filopodia experience maximum strain out of phase with other filopodia, resulting in some current injection that is out of phase with the action potentials. This effect is most easily observed during phases of the complex wave undergoing constructive interference, where inter-spike receptor potentials can be observed (**Fig. 5B**). These sub-threshold voltage deflections closely follow the stimulus waveform as reported in earlier electrophysiological studies [12]. During phases of destructive interference

action potentials are less likely to be triggered resulting in greater ISI values for spikes near this period of relative inactivity.

FR_{100} and ISI histograms of low and high d' comparisons, as predicted by the model, in select complex cases are presented in **Figure 6**. The discriminability of 140C and 150C by FR_{100} (**Fig. 6A**) is approximately 0.46, primarily due to the large overlap between the two underlying FR_{100} distributions and despite the low standard deviations of both distributions. Conversely, the d' value for the 110C and 190C waveforms by FR_{100}, which have more widely separated means, is 3.32 (**Fig. 6B**). **Figures 6C** and **6D** show the ISI histograms of the same simulated electrophysiology activity as in **6A** and **6B**, with d' values of 0.09 (**Fig. 6C**) and 0.32 (**Fig. 6D**).

Because firing rates were measured over periods of 100ms, spikes were averaged at too low of a rate to detect even the slowest beat frequency in the complex set (10 Hz) from the 110C stimulus, which has a period of 100ms. Equivalently, the sampling rate of FR_{100} (10 samples per second) is sub-Nyquist with respect to beat frequency– a FR_{50} code (20 samples per second) would be needed to detect the alternating periods of spiking activity and inactivity that a highly dissonant stimulus elicits in the simulated nerve. In comparison, the ISI coding scheme, which captures the instantaneous firing rate, produces non-normal ISI histograms, showing bimodality (**Fig. 6C**) and severe skewedness (**Fig. 6D**) as a result of its ability to encode the phasic spiking patterns the complex stimuli induced. The ISI distributions also overlapped more and had

higher standard deviations, resulting in overall lower d' values than the FR_{100}-based calculation.

Additional experiments were performed to assess the effect of adding noise to the system. Noise experiments were performed for all stimulus comparisons, although for visual simplicity only representative high (110 v. 160), middle (110 v. 140), and low (140 v. 150) d' pairs were plotted (**Fig. 7**). For the simple stimuli, FR_{100} showed almost no change in d' with noise, provided the SNR was kept above 0dB (**Fig. 7A**). Other conditions showed more sensitivity to noise. Complex stimuli assessed by the FR_{100} (**Fig. 7B**), as well as simple and complex stimuli d' measurements obtained from the ISI-based calculation (**Figs. 7C and 7D**) showed a slight drop in d' values as SNR went to 0dB. This indicated the greater noise sensitivity of the ISI calculation, and a greater susceptibility to noise that the complex condition elicited overall as compared to the simple stimulus set.

3.3 Comparison of the Model to Human Psychophysical Experiments

Discriminability measurements were made for all comparisons in both the simple and complex stimulus sets, based on the mean and standard deviations of coding scheme histograms (see **Figs 6A-4D** for examples). Next, these simulated d' values were compared to the corresponding d' values determined in the psychophysical setting. The results of these comparisons are shown in **Figure 8.** Simulated d' values were not found to correlate with psychophysical d'

values when simple and complex sets are grouped, regardless of the coding scheme used (**Fig 8A**). However, when the simple and complex stimulus sets were compared separately there was a significant ($p<0.01$) correlation between the psychophysical and simulated data when FR100, but not ISI was used to analyze the simple simulation data (**Fig 8B**). Additionally, in the complex case both ISI and FR100 were highly and significantly correlated with the psychophysical discriminabilites ($p<0.001$ in both cases) as shown in **Figure 8C**. The results in **Figure 8** were obtained in noiseless conditions.

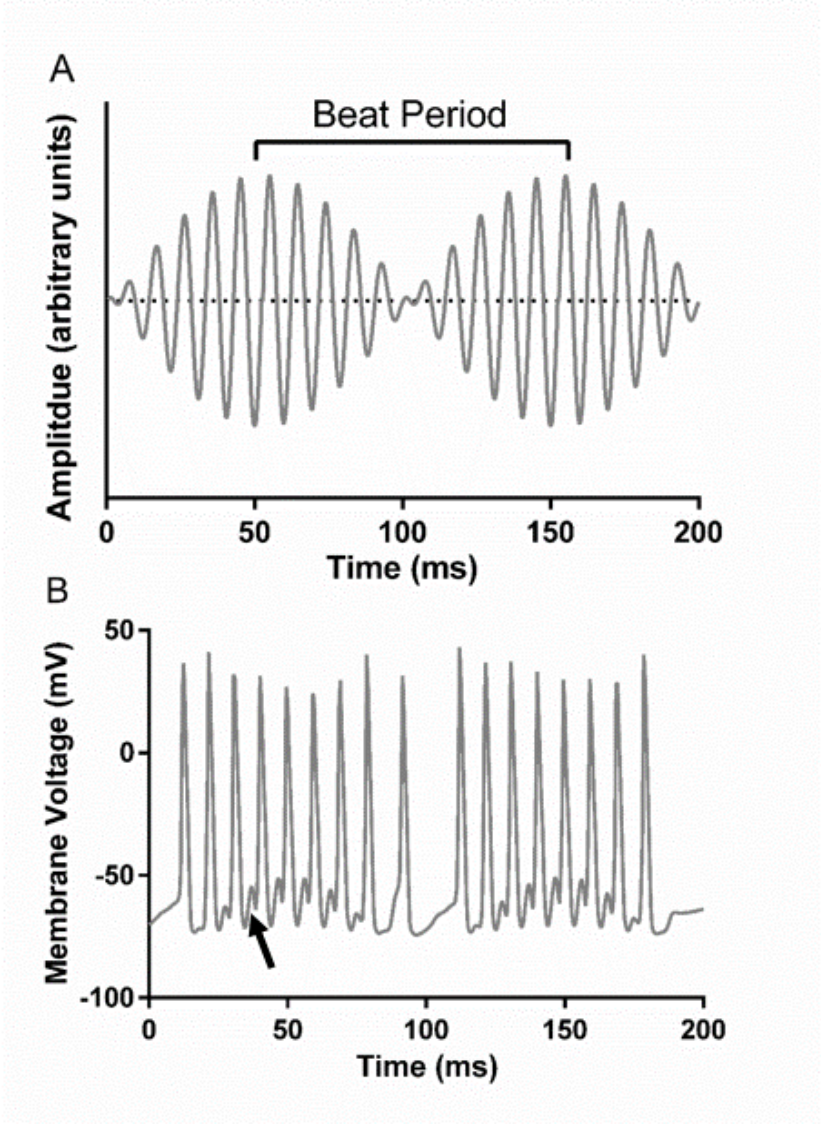

Figure 5. Example complex waveform and simulated response. (A) Two "beats" of the stimulus waveform for 110C are shown. **(B)** The simulated neuronal response to the stimulus in **(A)**. Both receptor potentials (peaks below the action potential threshold, indicated by the arrow) and action potentials are observed.

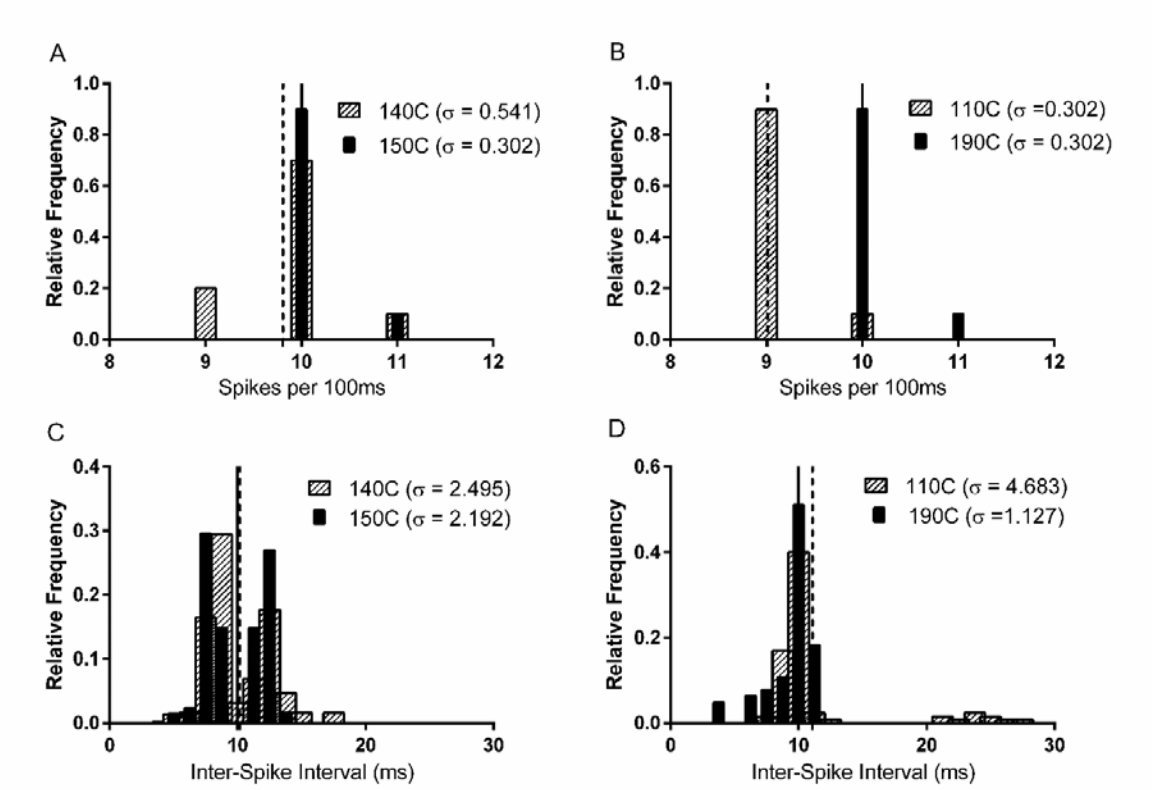

Figure 6. Histograms of simulated neuronal response. Vertical lines represent the means of the lower frequency stimulus (dashed line) and higher frequency stimulus (solid line). Standard deviations are listed in legend. **(A)** Low d' comparisons such as 140C versus 150C have heavily overlapping distributions for FR_{100}. **(B)** High d' comparisons have less overlapping distributions for the same coding scheme. **(C)** Likewise, ISI distributions that overlap more have lower discriminability than **(D)** those with less overlapping distributions and further separated means. The underlying ISI distributions **(C and D)** appear less normally distributed than their FR counterparts **(A and B)**. Many ISI distributions have long tails or appear bimodal, due to the presence of beats in the stimulus.

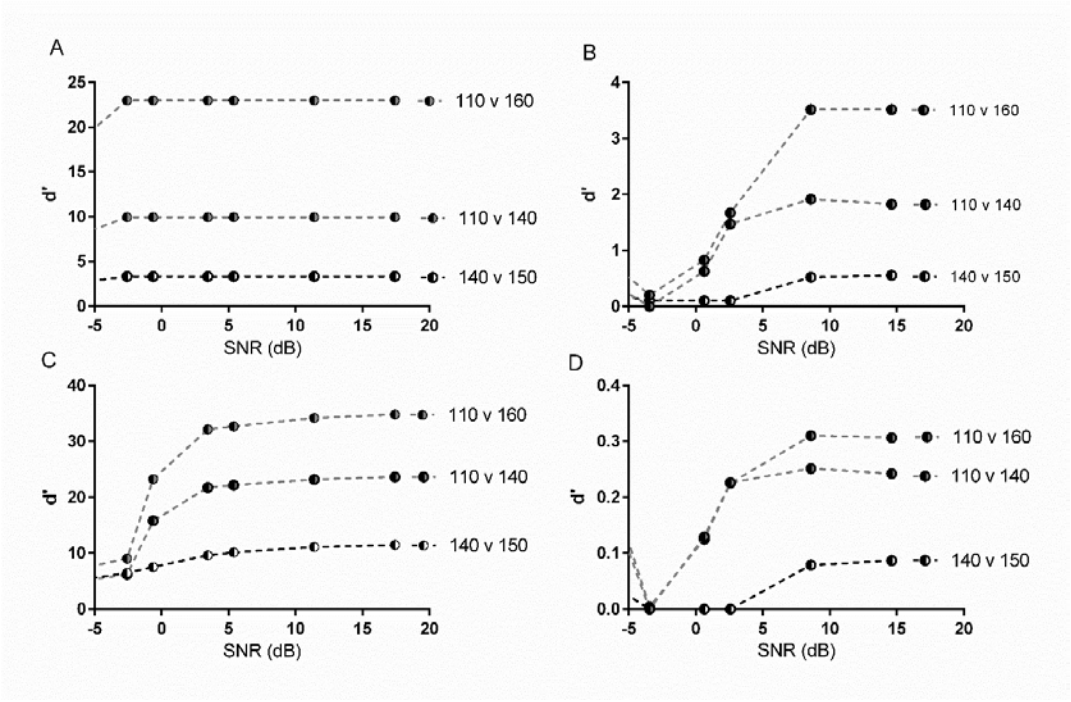

Figure 7. Effect of noise on select high, middle, and low d' comparisons. **(A)** Simple stimuli analyzed using a FR_{100} code are unaffected by noise above 0dB SNR. **(B)** Complex stimuli analyzed by FR_{100} are more sensitive to noise, showing d' drop-offs around 10dB. **(C)** Simple stimuli analyzed using ISI show a steady decline in d' as SNR decreases above 0dB, and a sharp drop-off below. **(D)** Complex stimuli analyzed by ISI are also affected by noise above 0dB. The vertical axis is scaled differently for each plot.

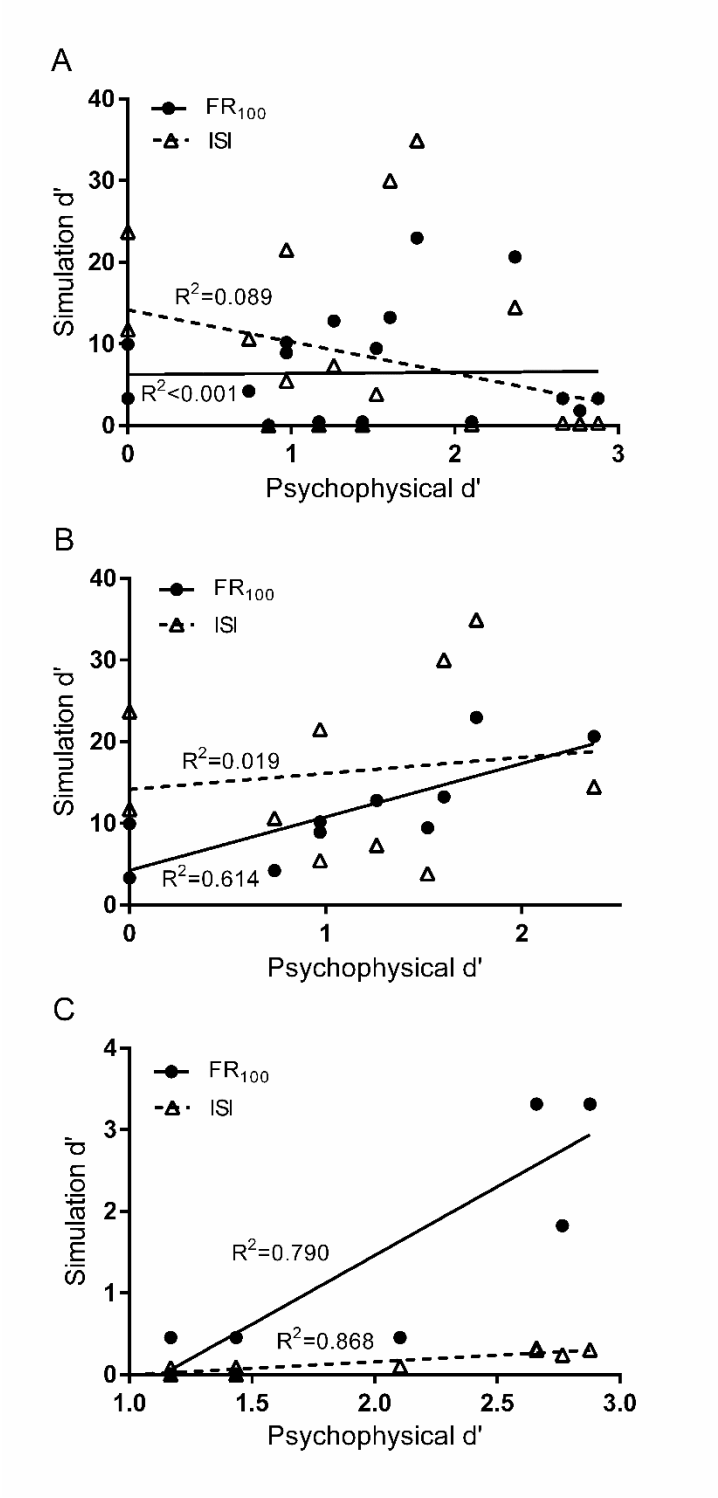

Figure 8. Psychophysically determined d' versus d' predicted by the model for all stimulus comparisons. Each dot represents a

frequency comparison, such as 110C versus 190C. **(A)** Both stimulus sets. **(B)** Simple only. **(C)** Complex only. While the simple **(B)** and complex **(C)** sets cannot be fit by the same line, the FR_{100} model data correlates linearly with psychophysical data when the two sets are analyzed separately **(B and C)**. Unlike FR_{100}, the average ISI decoding scheme only produces a good fit when used to analyze complex data **(B and C)**.

4 Discussion

The goal of this study was to simulate psychophysical discriminability of haptic vibrations using a detailed mechanistic model of a single PC; that is, to use a computer simulation of a PC to predict our ability to distinguish different touch sensations. Many earlier experimental approaches have paired *in vivo* electrophysiology recordings with psychophysical experiments, an approach emulated here. To assess the computational-psychophysical approach, parallel psychophysical experiments were performed on human subjects and with a computer model over two sets of stimuli, one consisting of a single pure sinusoid component (simple), and the other, two components (complex). In this last chapter, the results of the psychophysical experiments, the simulation studies, and their comparison are discussed. Possible future experiments as well as implications of the results are also considered.

4.1 Discussion of Psychophysical Experiments

The psychophysical experiments relied on a fairly large vibrator (10 mm diameter), which unquestionably stimulated multiple PCs on the fingertip. Also, the subjects were free to move their finger during the test, so there may have been some variability as a result. While some subjects had higher or lower detection thresholds than others, stimuli were normalized to their individual thresholds. The most notable observation from the psychophysical experiments was that, in nine of the ten comparisons, the addition of a 100 Hz wave to both

stimuli resulted in significantly greater discriminability than was seen for the pure tones (**Fig. 4D**). In a paired t-test of the pooled d' scores, the difference between simple and complex stimuli were on average nonzero, showing a significantly ($p < 0.01$) improved d' value for complex stimuli over their simple counterparts. Certainly, some subjects were better at the discrimination task than others, but consistently, on an individual basis as well, subjects were better at discriminating complex stimuli.

The mechanism for this effect is not immediately clear, but we observe that the effect was greater when the frequencies being compared were closer to 100 Hz, which would result in a lower beat frequency (i.e., a greater degree of dissonance) and perhaps a more recognizable signal. In other words, the difference between beat frequencies of 10 Hz and 40 Hz may be more detectable than the difference between pure tones of 110 Hz and 140 Hz. An earlier study on consonance and dissonance perception of vibrotactile chords [32] concluded that the "dominant sensory cue" for dissonance is beat frequency, regardless of the base frequency used. Our results support the finding that dissonant haptic vibrations are recognizable, and moreover that the presence of beat frequencies improves the discriminability of such stimuli.

4.2 Discussion of Simulation Experiments

Earlier studies [2] have suggested that the PC has a strong tendency to phase-lock its spike rate with single-component stimulus frequencies at supra-

threshold levels which is suggestive of a rate code, where firing rate indicates the frequency of the stimulus and overall recruitment in the population captures stimulus intensity. Other work [9] has concluded that stimulus information is derived from the mean inter-spike interval (ISI) of afferent cells, a finding that is not necessarily at odds with the rate code hypothesis. We found that these very simple encoding schemes based on FR_{100} or ISI were able to discriminate between pairs of complex tones, and the predicted discriminability correlated well with that observed in the experiments. However the ISI model did not correlate with the psychophysical data for simple stimuli, in part because the ISI distribution was highly non-normal, in violation of the fundamental assumptions defining the calculation of d'. Non-normal ISI histograms are especially evident for the complex stimuli (**Figs. 6C and 6D**). Perhaps more importantly, after the first spike, the simulated neuron was not always able to repolarize quickly enough to produce a second spike phase-locked to the stimulus, leading to non-correlated ISI-based d' calculation, even for some noiseless simple stimuli.

A single action potential is sufficient to cause an increase in the action potential threshold, [33] resulting in a re-equilibration of electro-chemical membrane conditions where, presumably, phase-locked spiking can be restored, albeit with action potentials waveforms that altered slightly (higher thresholds, attenuated peak heights). This effect can be thought of as a form of noise, perhaps intrinsic to the underlying biology, and certainly to the model. The FR coding scheme, unlike ISI, may be smoothing this “biological” noise.

Our results suggest that firing rate is a more reliable decoding scheme than ISI, but it does not discount the ISI code, especially for a system that includes multiple cells or post-synaptic neurons. The crude ISI coding scheme used in our study was not consistent with how PC output is processed. It was not that ISI does not provide a useful measure; instead, the variability of the calculated ISI leads to d' values that are not consistent with the psychophysical data. ISI may well contribute to the information content of the signal, but the hypothesis that the mean ISI in a single PC, per se, encodes vibrotactile signals is rejected.

Previously, Bensmaia *et al.* [13] have argued that small differences in the frequency-response curves of individual PCs (due to variations in PC structure, such as the number of lamellae, and/or the positions of individual PCs relative to the stimulus source) allow PC populations to convey information about multiple spectral components in parallel, through what the authors termed "quasi-independent mini-channels." In other words, Bensmaia argued that multi-component stimuli are distinguishable at the population level even if they may not be distinguishable by a single PC. Clearly, the phenomenon merits further investigation, especially considering theoretical studies demonstrating effects of PC structure on function [21] [22] [23].

While such mini-channels are likely integral to distinguishing more nuanced complex waves (for instance complex waves with more than two components, or harmonics), it is clear from this study that only a single PC is

necessary to model the qualitative response to highly dissonant two-component waves. Interestingly, the natural variability imposed by a mini-channel model may have another important role: adding variance to the population response to a single frequency stimulus. The psychophysical discriminability of single-frequency stimuli were over-predicted by the single-PC model used here, even at relatively high noise (-5dB SNR). This is compared to the single cell response to complex waveforms, which were modeled accurately using a FR_{100} decoding paradigm.

4.3 Discussion of Comparison

The FR_{100} *in silico* experiments correlated with the psychophysical experiments for both pure and complex stimuli individually, but not when viewed together. That is, if one pure-wave comparison (e.g., 110S versus 190S) was more discriminable than another psychophysically (e.g., 140S versus 150S), the *in silico* d' values for the more discriminable comparison also tended to be larger, and likewise for the complex stimuli. If, however, a certain pair of complex stimuli was more discriminable than a certain pair of simple stimuli, the *in silico* value for the complex comparison was not necessarily larger than that of the simple comparison. Again, the meaning of this result is not clear. It suggests that certain aspects of discriminability can be captured by the FR_{100} model, but that there are other factors involved that become particularly important when the type of stimulus is changed. This may stem from population effects that were not

being modeled in this study, or from the over-simplicity of the decoding methodology used to calculate d'.

The *in silico* experiments generally yielded higher d's than corresponding psychophysical experiments for the pure stimuli, even at moderate noise (SNR 0 dB). In contrast, the model tended to under predict d' for the complex stimuli. This result is consistent with Bensmaia's contention [13] that the more complex the signal, the more important the population encoding becomes - the single-PC model discriminated pure signals better, but for a complex signal, a single PC would be insufficient and thus the single-PC model under predicts psychophysical performance.

4.4 Implications and Future Directions

Further investigations should explore alternative encoding / discrimination schemes (e.g., van Rossum spike distance [34]) as well as the possibility of a population code. A complete understanding of PC coding depends on more advanced questions of PC population behavior such as how their activity is summed to produce meaningful sensations, something the model used in this study is currently unable to do. Future studies should therefor incorporate populations of receptors. Such a study would be able to assess in more detail the "mini-channel" hypothesis, and how variably tuned PC populations encode different stimulus types (i.e. complex or simple). Understanding somatosensation as a whole poses a greater challenge still, where touch

information is encoded by multiple specialized afferents (PCs, MCs, and slowly adapting mechanoreceptors) and used downstream. In principle, all these components could be modeled, and included in future studies that take on a paired psychophysical-computational approach.

How each piece of this multiscale model would be tuned, to physiology and to pathology, will surely require a multitude of validation approaches, beyond threshold or discriminability measurements. What a paired psychophysical-computational approach suggests, however, is that the multiphysics model is also a behavioral model. That is, a single PC modeled in such detail is sufficient to recapitulate some of the observed response of the whole organism to a simple behavioral assessment. This may provide opportunities to evaluate the detailed changes in PC physiology associated with both gerontology and pathology, as well as a more systematic investigation of its role in the neural representation of more natural haptic sensations across populations of receptors.

Dupuytren's contracture, as previously alluded to, is an example of specific disease state involving the receptor [24]. PCs in Dupuytren's patients have been found to be larger and more layered, and while it has been suggested that Dupuytren's nodules are PC-derived, the disease etiology is not known [25]. The combined psychophysical-simulation model that is the subject of this study could be applied to pathology studies of the PC's role in Dupuytren's. For instance, if the relationship between disease progression and PC behavior can be understood, this could provide a basis for the development of at-home patient

monitoring systems. If, in another case, changes in PC structure are not merely epiphenomenal but casual to Dupuytren's, therapy targets could possibly be identified using the modeling approach.

More generally, these findings apply to the design of better haptic technology, where it has been previously determined that dissonant vibrotactile stimuli can be used to diversify modes of a haptic device [32], but measures of the psychophysical discriminability of these stimuli have not yet been reported, nor have the root biological causes. Additionally, under the broader umbrella of haptics, in the growing body of neuro-prosthetic research, a paired psychophysical-computational methodology may be most relevant for the design of somatosensory prosthetics and bio-mimetic algorithms, as mentioned previously, where a detailed looked at the biological mechanisms driving the psychophysical response is essential.

Conclusion

These psychophysical experiments suggest that making a pure tone dissonant by adding a nearby frequency increases its discriminability against more consonant (higher beat frequency) comparisons. The paired psychophysical-computational analysis shows that the neural response of a single simulated PC correlates to corresponding psychophysical data, demonstrating for the first time that a multiscale, multiphysics model of this receptor can be used in place of an *in vivo* paradigm to study the physiology of the receptor. FR_{100} was found to be a better decoding methodology in this setting than ISI, since it is less susceptible to low SNR settings and because it better fits the underlying assumptions of d'.

References

[1] B. Stark, T. Carlstedt, G. Hallin and M. Risling, "Distribution of human Pacinian corpuscles in the hand," *Journal of Hand Surgery,* vol. 23B, no. 3, pp. 370-372, 1998.

[2] K. O. Johnson, "The roles and functions of cutaneous mechanoreceptors," *Current Opinion in Neurobiology,* vol. 11, pp. 455-461, 2001.

[3] S. Bentivoglio and P. Pacini, "Fillipo Pacini: A determined observer," *Brain Research Bulletin,* vol. 38, no. 2, pp. 161-165, 1995.

[4] H. C and et al, "Vibrotactile presentationof musical notes to the glabrous skin for adults with normal hearing or a hearing impairment: thresholds, dynamic range and high-frequency perception," *PLoS ONE,* vol. 11, no. 5, p. e0155807, 2016.

[5] S. Bensmaia and L. Miller, "Restoring sensorimotor function through intracotrical interfaces: progress and looming challenges," *Nature Reviews Neuroscience,* vol. 15, pp. 313-325, 2014.

[6] B. P. Delhaye, E. W. Schulter and S. J. Bensmaia, "Robo-psychophysics: Extracting behaviorally relevant features from the output of sensors on a prosthetic finger," *IEEE Transactions on Haptics,* vol. 9, no. 4, pp. 499-507, 2016.

[7] L. Osborn, R. R. Kaliki, A. B. Soares and N. V. Thakor, "Neuromimetic event-based detectionfor closed-loop tactile feedback control of upper limb prostheses," *IEEE Transactions on Haptics,* vol. 9, no. 2, pp. 196-206, 2016.

[8] N. Patel, J. Jankovic and M. Hallet, "Sensory aspects of movement disorers," *Lancet Neurology,* vol. 13, no. 1, pp. 100-112, 2014.

[9] K. Horch, "Coding of vibrotactile stimulus frequency by Pacinian corpuscle afferents," *J. Acoustical Society of America,* vol. 89, pp. 2827-2836, 1991.

[10] V. B. Mountcastle, R. H. Lamotte and C. Giancarlo, "Detection thresholds for stimuli in humans and monkeys: comparison with threshold events in mechanoreceptive afferent nerve fibers innervating the monkey hand," *Journal of Neurophysiology,* vol. 35, no. 1, pp. 122-136, 1972.

[11] R. T. Varrillo, "Age related changes in the sensitivity to vibration," *Journal of Gerontology,* vol. 35, no. 2, pp. 185-193, 1980.

[12] S. J. Bolanowski and J. J. Zwislocki, "Intensity and frequency characteristics of Pacinian corpuscles. II. Receptor potentials," *Jounral of Neurophysiology,* vol. 51, no. 4, pp. 812-830, 1986.

[13] S. M. Bensmaia, M. Hollins and J. Yau, "Vibrotactile intensity and frequency information in the Pacinian system: a psychophysical model," *Perception and Psychophysics,* vol. 67, no. 5, pp. 828-841, 2005.

[14] J. C. Quindlen, H. K. Stolarski, M. D. Johnson and V. H. Barocas, "A multiscale model of the Pacinian corpuscle," *Integrative Biology,* vol. 8, pp. 1111-1125, 2016.

[15] N. Cuana and G. Manna, "The structure of human digital Pacinian corpuscles (Corpuscula Lamellosa) and its functional significance," *J Anat,* vol. 92, no. 4, pp. 1-20, 1958.

[16] D. Pease and A. Quilliam, "Electron microscopy of the Pacinian corpuscle," *J. Biophysic. and Biochem. Cytol.,* vol. 3, no. 3, pp. 331-342, 1957.

[17] D. C. Pease and T. A. Quilliam, "Electron microscopy of the Pacinian corpuscle," *J Biophys Biochem Cytol,* vol. 3, no. 3, pp. 331-342, 1957.

[18] J. Zelena, "The development of Pacinian corpuscles," *Jounral of Neurocytology,* vol. 7, no. 1, pp. 71-91, 1978.

[19] A. Zimmerman, L. Bai and D. D. Ginty, "The gentle touch receptors of mammalian skin," *Science,* vol. 346, no. 6212, pp. 950-954, 2014.

[20] L. Pawson, N. B. Slepecky and S. J. Bolanowski, "Immunocytochemical identification of proteins within the Pacinian corpuscle," *Somatosens Mot Res,* vol. 17, no. 2, pp. 159-170, 2000.

[21] W. R. Loewenstein and R. Skalak, "Mechanical transmission in a Pacinian corpuscle. An analysis and a theory," *J. Physiol.,* vol. 182, pp. 346-378, 1966.

[22] M. H. Holmes and J. Bell, "A model of a sensory mechanoreceptor derived from homogenization," *Society for Industrial and Applied Mathematics,* vol. 50, no. 1, pp. 147-166, 1990.

[23] J. C. Quindlen, B. Guclu, E. A. Schepis and V. H. Barocas, "Computational parametric analysis of the mechanical response of structurally varying Pacinian Corpuscles," *J. Biomechanical Engineering,* p. doi:10.1115/1.4036603, 2017.

[24] N. Akyurek, O. Ataoglu, S. Cenetoglu, S. Ozmen, T. Cavusoglu and R. Yavuzer, "Pacinian corpuscle hyperplasia coexisting with Dupuytren's contracture," *Ann Plast Surg,* vol. 45, no. 2, pp. 220-222, 2000.

[25] W. R. Ehrmantant, W. P. Graham, J. Twofight, D. R. Mackay and H. P. Ehrlich, "A histological and anatomical profile of Pacinian corpuscles from Dupuytren's contracture and the expression of nerve growth factor," *Plast Reconstr Surg,* vol. 114, no. 3, pp. 721-727, 2004.

[26] D. F. Eberl, R. W. Hardy and M. J. Kerman, "Genetically similar transduction mechanisms for touch and hearing in drosophila," *J Neuroscience,* vol. 20, no. 16, pp. 5981-5988, 2000.

[27] M. Heidenreich, S. G. Lechner, V. Vardanyan, C. Wetzel, C. W. Cremers, E. M. De Lenheer, G. Aranguez, M. A. Moreno-Pelayo, T. J. Jentsch and G. R. Lewin, "KCNQ4 K+ channels tune mechanoreceptors for nomral touch sensation in mous and man," *Nature Neuroscience,* vol. 15, p. 15, 2012.

[28] H. P. Saal, X. Wang and S. J. Bensmaia, "Importance of spike timing in touch: an analogy with hearing?," *Current Opinion in Neurobiology,* vol. 40, pp. 142-149, 2016.

[29] N. A. Macmillan and C. D. Creelman, Detection theory, a user's guide, 2nd ed. ed., Mahwah: Lawrence Erlbaum Associates, Inc., 2005.

[30] Syscom Design, "WGM-201 Waveform Generator Manual," 2012. [Online]. Available: http://www.syscompdesign.com/assets/images/Manuals/wgm-201-manual-108.pdf. [Accessed October 2016].

[31] H. Levitt, "Transformed up-down methods in psychoacoustics," *The Journal of the Acoustical Society of America*, vol. 49, no. 2, pp. 467-477, 1970.

[32] Y. Yoo, I. Hwang and S. Choi, "Consonance of vibrotactile chords," *IEEE Transaction on Haptics*, vol. 7, no. 1, pp. 3-13, 2014.

[33] D. A. Henze and G. Buzsaki, "Action potential threshold of hippocampal pyramidal cells in vivo is increased by recent spiking activity," *Neuroscience*, vol. 105, no. 1, pp. 121-130, 2001.

[34] M. C. W. van Rossum, "A Novel Spike Distance," *Neural Computation*, vol. 13, no. 4, pp. 751-763, 2001.

Appendices

Appendix A

turn_on.m

```
%% Turn on waveform generator and initiate settings
% output_info = instrhwinfo('serial')
s = serial('/dev/cu.usbserial-WG0KJ2SO'); %ensure this usbserial is
correct for the machine (computer) you are using!
set(s,'BaudRate',230400);
set(s,'FlowControl','Hardware');
set(s,'Terminator','CR/LF');
fopen(s);
```

enable_buzzer.m

```
%%Turn buzzer back on after stopping with <disable_buzzer.m>
%will turn back on to parameter you had it at unless told otherwise
fprintf(s, '%s\n', 'E');
```

set_amplitude.m

```
%% Signal Generator Code to Set Amplitude
% JCQ 6-16-16
% This function takes the input of amplitude in V and sets the
% signal generator to run at that amplitude
function set_amplitude(s,amp)
    %Amplitude is input in V
    % Amplitude scale = nnn is an 8 bit ascii number ranging from 0 -
255
    % output amplitude = (nnn/255) * 10 V
        amp_scaled = floor((amp/10)*255);
        amp_output = sprintf('A %d',amp_scaled);
        fprintf(s, '%s\n', amp_output);
        clear amp_scaled amp amp_output;
end
```

set_frequency.m

```
%% Signal Generator Code to Set Frequency
% JCQ 6-16-16
% This function takes the input of frequency in Hz and sets the
% signal generator to run at that frequency
function set_frequency(s,freq)
    %Frequency is input in Hz
    % Frequency = use F aaa bbb ccc ddd where aaa through ddd are 8 bit
    % ascii numbers that make up a 32-bit number (YYYY) to set the
output
    % frequency
    % output frequency = YYYY*0.031247735 Hz
        freq_scaled = floor(freq/0.031247735); %Freq in Counts
        %Convert this to a hexadecimal number
```

```
        freq_hex = dec2hex(freq_scaled);
        %Group the hex number by pairs starting at the right end.
        length_hex = length(freq_hex);
        hex_backwards = fliplr(freq_hex);
        pairs = cell(1,4);
        for n_hex = 1:round(length_hex/2)
            if 2*n_hex <= length_hex
                pairs{n_hex} = fliplr(hex_backwards(2*n_hex-
1:2*n_hex));
            else pairs{n_hex} = hex_backwards(2*n_hex-1);
            end
        end
        %fill in with zeros if doesn't fill all 4
        if round(length_hex/2)<4
            for n_fill = round(length_hex/2)+1:4
                pairs{n_fill} = 0;
            end
        end
        pairs = fliplr(pairs);
        %Convert each pair to the corresponding decimal number,
        %which will range from 0 to 255
        dec = zeros(1,4);
        for n_pairs = 1:4
            dec(n_pairs) = hex2dec(pairs{n_pairs});
        end
        %Format into string
        dec_string = cell(1,4);
        for n_pairs = 1:4
            if n_pairs < 4
                temp = sprintf('%d ',(dec(n_pairs))); %add in space
            else temp = sprintf('%d',(dec(n_pairs)));
            end
            dec_string{n_pairs}=temp; clear temp;
        end
        freq_string = [];
        for n_pairs = 1:4
            freq_string = horzcat(freq_string,dec_string{n_pairs});
        end
        freq_output = sprintf('F %s',freq_string);
        fprintf(s, '%s\n', freq_output);
        clear n_pairs dec_string pairs dec freq_hex freq_scaled freq
...
            length_hex n_hex hex_backwards n_fill freq_string
freq_output
end
```

load_custom.m

```
%% Signal Generator Code to Load Arbitrary Function
% JCQ 7-12-16
% This function takes the input of a text file (of arbitrary function),
% input of amplitude, and pause time
function load_custom(s,load_file,amp,pause_time)
fprintf(s, '%s\n', 'A 0');
%load_file= sprintf('sine.dat');
```

```
if iscellstr(load_file) == 1
data_file = load(load_file); clear load_file;
else
data_file = load_file;
end
for n = 1:length(data_file)
    clear address data print_statement;
    address = n - 1;
    data = data_file(n);
    print_statement = sprintf('S %d %d',address,data);
    fprintf(s, '%s\n', print_statement);
end
pause(pause_time)
set_amplitude(s,amp)
    %fprintf(s, '%s\n', 'A 120');
end
```

disable_buzzer.m

```
%Code to stop the buzzer - need to turn back on using <enable_buzzer.m>
fprintf(s, '%s\n', 'e');
```

shut_down.m

```
%% End the serial port session
fclose(s)
delete(s)
clear s
```

Appendix B

up_down_threhsolding.m

```
%% up/down thresholding
%NDT 2016
%For reference see: Transformed Up-Down Methods in Psychoacoustics,
%       by H. Levitt, for J. Accoustical Society of America, 1970,
p.467-477
%This script will only run when the USB outputs to a signal generator!
Preferably SysCom WGM-201
%This script requires user input while it runs!
%Recommendation: save the workplace variables that are output from
script when finished running
%if script crashes after 'turn_on;' command but before 'shut_down;'
then you must
%manually run shut_down; before you re-run thresholder2!!
close all; clc;
%THE FOLLOWING VARIABLES MAY NEED TO BE ADJUSTED PRIOR TO RUNNING
SCRIPT
%This script will test the following frequencies, sin waves only!
frqs = [100,110,140,150,160,190]; %<== CHANGE: frequencies to test
%FIRST MUST BE LOWEST
x50 = [2.5,2.5,2.5,2.5,2.5,2.5]; %<== CHANGE: guess at threshold for
each frequency to test (x50 must have sam indexing as frqs)
stepsize = [.5,.5,.5,.5,.5,.5]; %<== CHANGE: enter stepsize (stepsize
and frqs must have the same indexing as well!)
%specificy timing parameters
runs = 3; %<== CHANGE: since this is up/down thresholding: specify how
many total up and down cycles to run
time_on = 2; %<== CHANGE: how long to give the stimulus
time_off = 1; %<== CHANGE: how long to pause after with no stimulus
%%  Turn on the signal generator!!!
turn_on;
%equipt sin wave and enable buzzer output
fprintf(s,'%s\n','W 0'); %sine wave
enable_buzzer;
%% Run vibrations
for frq_n = 1:size(frqs,2); %for every frequency specified
    amp_n = x50(frq_n); %start at the x50 amplitude (the estimated
threshold)
    half = 1; %this comes into play later on, and is only used once per
frequency
    Fth = []; %initialize the temporary FrequencyScore matrix
(precurser to 'FrequencyScore')
    updown = 1; %positive means going up, but it will flip right away
    for r = 1:runs
        if r > runs/2 && half == 1 %this halves the stepsize once
halfway through the number of runs
            stepsize(frq_n) = stepsize(frq_n)/2; %halve the stepsize,
as recommended by H Levitt.
            half = 0; %so it only happens once!
        end
```

```
    updown = -updown; %at the beginning of every run, flip the updown
direction
    threshlimit = 0; %this is how the program nows when to terminate
the run
       if updown < 0  %if it is on a 'down' cycle
                 while threshlimit == 0; %for as long the subject says
he/she can feel the stimulus
                     amp_n = amp_n + (updown*stepsize(frq_n)); %decrease
the amplitude (since updown<0)
                     if amp_n < 0
                        amp_n =0; %makes sure it doesn't try to output a
negative ampliude
                     end
                     set_frequency(s,frqs(frq_n));
                     set_amplitude(s,amp_n);
                     %disp(amp_n) %for troubleshooting
                     pause(time_on); %wait
                         prompt = 'Can you feel that? 9=YES / 0=NO
...then press ENTER ';
                         x = input(prompt);
                              if x == 9;
                                  Fthx = 1; %value to be stored
                              else
                                  Fthx = 0; %value to be stored
                                  disp('Threshold reached!')
                                  threshlimit = 1; %stimulus has
decreased below the limit of detection => terminate while loop
                              end
                 Fth = [Fth; [amp_n,Fthx]]; %store outcome
                 set_amplitude(s,0) %silence the buzzer
                 pause(time_off); %wait
                 end
       else %the other possibility is that updown = 1 i.e. updown>0
                 while threshlimit == 0; %for as long the subject says
he/she cannot feel the stimulus
                 amp_n = amp_n + (updown*stepsize(frq_n));%increase the
amplitude (since updown>0)
                      if amp_n > 10
                            amp_n =10; %makes sure it doesn't try to
output an amplitude beyond its limit of 10
                      end
                 set_amplitude(s,amp_n);
                 set_frequency(s,frqs(frq_n));
                 %disp(amp_n) %for troubleshooting
                 pause(time_on); %wait
                     prompt = 'Can you feel that? 9=YES / 0=NO ...then
press ENTER ';
                     x = input(prompt);
                          if x == 9;
                              disp('YES. Good job!')
                              Fthx = 1; %value to be stored
                              threshlimit = 1; %stimulus has increased
above the limit of detection => terminate while loop
                          else
```

```
                            Fthx = 0; %value to be stored
                        end
                Fth = [Fth; [amp_n,Fthx]]; %store outcome
                set_amplitude(s,0) %silence the buzzer
                pause(time_off); %wait
                end
        end
    end
    eval(sprintf('FrequencyScore%d = Fth', frqs(frq_n))); %store the
outcome of all the runs for the frequency, namning the variable
accordingly
end
%% Close down the serial port (terminate Matlab's USB-mediated
connection to signal generator)
shut_down; %if the script already ran turn_on; you must run this
section manually before you rerun the script!  If you don't you need to
re-start Matlab.
%% Make plot of results and calculate the frequency-specific thresholds
Threshold_results = [];
for frq_n = 1:size(frqs,2) %this loops back through the frequencies,
after all data is collected
a =eval(sprintf('FrequencyScore%d', frqs(1,frq_n))); %a is essentially
serving a similar function as Fth did.
FrequencyScore=a;
%plotting results
figure;
sz = size(FrequencyScore,1);
X = linspace(1,sz,sz);
scatter(X,FrequencyScore(:,1))
hold on
for f = 1:sz
   if (FrequencyScore(f,2)) == 1
      scatter(X(f),FrequencyScore(f,1),'*r')
   end
end
%namef = whos('FrequencyScore');
%title(namef.name);
xlabel('Sample Number');
ylabel('Amplitude (no units)');
%more plotting
ma =[]; %initiate an empty list to later find the mean threshold value
    for i = 1:size(a,1)-1
       if a(i,2) == 0 && a(i+1,2) == 1 %threshold occurs somewhere
between the unfelt and felt stimuli...
        ma=[ma;a(i,1)];  %so take the middle point and append it to the
list
       elseif a(i,2) == 1 && a(i+1,2) == 0 %threshold occurs somewhere
between the felt and unfelt stimuli...
        ma=[ma;a(i,1)]; %so take the middle point and append it to the
list
       end
    end
    thresholda = mean(ma); %the threshold of a ('thresholda') is
calculated
```

```
    eval(sprintf('Threshold_%d=thresholda', frqs(1,frq_n))) %then this
value is stored in a variable named using the frequency vector
    Threshold_results = [Threshold_results;[frqs(frq_n),thresholda]];
    thrshline=repmat(thresholda,size(a,1),1); %create a line
representing the mean calculated threshold value
    hold on
    plot(thrshline,'r'); %then plot this line
end
%% Clean up workspace - for troubleshooting, comment this section out.
%remove all unneeded variables from workspace:
clear Fth Fthx a amp_n frq_n half i ma prompt r runs stepsize
threshlimit thresholda thrshline time_off time_on updown x x50;
```

Appendix C

same_different_experiment.m

```
%%Same-Difference-Experiment (Detection Theory)
%NDT 2016
%As described in: N. A. Macmillan and C. D. Creelman, Detection theory,
a user's guide, 2nd ed. ed., Mahwah: Lawrence Erlbaum Associates, Inc.,
2005.
%Note, this version only tests complex waves.
%To make simple waves, see line commented with:  %FOR SIMPLE WAVES
%Naming conventions should also be changed for simple wave experiments
close all; clc
%outputs SRMs (Stimulus Response Matrices) where left column means the
user
%marked the two stimili as "different" and the right column for "same"
input('press enter to begin');
repetition = 2;
%In Threshold_results, the first entry must be the lowest (base)
frequency
TR = Threshold_results; %this matrix is to be imported from
'thresholder2.m'
Nt = ((size(TR,1)-1)^2); % repetition; %number of trials;
%thresholds     %each row is a different frequency, the first column
states
%that frequency, the second column show the threshold of that frequency
Nf = size(TR,1); %number of frequencies
Trial_Index_A = repmat(2:Nf, [1 round(length(2:Nf))]); %start at 2 so
that the base frequency is not included
%Trial_Index_B = randi([2 Nf],Nt,1);
next=0;
for iter=1:round(Nt/length(2:Nf))
    j=0;
    i=1;
    while i < Nf
        Trial_Index_B(i+next) = Trial_Index_A(iter+j);
        j=j+1; i=i+1;
    end
    next=next+(Nf-1);
end
Trial_Index_B = Trial_Index_B(1:length(Trial_Index_A));
Trial_Index_AB = [Trial_Index_A; Trial_Index_B];
Trial_Index_AB_repetition = repmat(Trial_Index_AB, repetition);
Trial_Index_AB_repetition = Trial_Index_AB_repetition(1:2,:)';
Trial_Index_AB_shuffled = ...

Trial_Index_AB_repetition(randperm(size(Trial_Index_AB_repetition,1)),:
);
Thrmax = max(TR);%set everything relative to the highest threshold
frequency...
amp = 4*Thrmax(1,2); %...and use an amplitude 4X this highest threshold
if amp > 10 %make sure it is within limits tho
   amp = 10;
```

```
end
%% From the Trial_Index...
turn_on;
enable_buzzer;
for Q=1:size(Trial_Index_AB_shuffled,1);
SD = 0; %refresh memory unit: this will go unchanged if stimuli are
marked as same
FqA = Trial_Index_AB_shuffled(Q,1); %get the associated random
frequency
FqB = Trial_Index_AB_shuffled(Q,2);   %get the associated random
frequency
F2=TR(FqA,1); %for troubleshooting, delete later
F3=TR(FqB,1); %for troubleshooting, delete later
f1 = 1;
f2 = TR(FqA,1)/TR(1,1);
f2 = round(f2*10)/10;
f3 = TR(FqB,1)/TR(1,1);
f3 = round(f3*10)/10;
%NOTE: SRM stands for "Stimulus Response Matrix"
if f2==f3 %if the two frequencies are the same
    if exist(sprintf('SRM_%d_%d',TR(FqA,1),TR(FqB,1))) == 0 %and if an
SRM doesn't exist
       evalc(sprintf('SRM_%d_%d=zeros(1,2);',TR(FqA,1),TR(FqB,1)))
%create a SRM for them
    end
elseif f2<f3 %if f2 is smaller than f3
    if exist(sprintf('SRM_%d_%d',TR(FqA,1),TR(FqB,1))) == 0 %and if an
SRM doesn't exist
       evalc(sprintf('SRM_%d_%d=zeros(1,2);',TR(FqA,1),TR(FqB,1)))
%create a SRM for them
    end
elseif f2>f3 %f2 is larger than f3
    if exist(sprintf('SRM_%d_%d',TR(FqB,1),TR(FqA,1))) == 0 %if the SRM
doesn't exist
       evalc(sprintf('SRM_%d_%d=zeros(1,2);',TR(FqB,1),TR(FqA,1)))
%create it
    end  %note: the elseif and else statements here ensure that the
lower frequency is always first in the SRM name.
end
%%
%common =lcm(sym1,sym2)-1 %this determines how many phases will be
needed until they synch up again
common = 10; %set at 10 in order to compare things with a ration of
1.1, 1.2, 1.3, ..., 1.9
%I believe if this is set to 100, the fastest frequency that can be
overlaid with 100 hz
%is 128hz, since 256/2 = 128, since 100hz played 100 cycles is 1 second
so
%256 samples per second => max frequency of 128 by Nyquist Theorem
%but how many cycles does the longest freqeucny undergo?
%because this will need to be factored into the runs, as a
correction...
signal_length = 2*pi*common;  %safety feature?  so it doesn't
overshoot?
```

```
t = linspace(0,signal_length,256); %create 256 points between 0 and
2*pi*common...
q = linspace(1,256,256);
%t=double(t); %go from symbolic to double
y =  128*sin(t);
y2 = 128*sin(f2*t);
y3 = 128*sin(f3*t);
yA = y+y2; %yA = y2; %FOR SIMPLE WAVES
yB = y+y3; %yB = y3; %FOR SIMPLE WAVES
yA=yA';
normvalue = max(abs(yA));
yA = yA/normvalue;
yA = 128*yA;
yA = ceil(yA);
yA = yA+127;
max(yA); %for troubleshooting
min(yA); %for troubleshooting
yB=yB';
normvalue = max(abs(yB));
yB = yB/normvalue;
yB = 128*yB;
yB = ceil(yB);
yB = yB+127;
max(yB); %for troubleshooting
min(yB); %for troubleshooting
%FOR VISUALING
%  figure;
%  plot(q,yA','r');
%  hold on
%  plot(q,yB,'k');
% Practice Run 2
pause_time=1;
load_custom(s,yA,amp,pause_time);
set_frequency(s,10) %this actually is 100Hz, but since the thing
repeats 10 times
disp('Now Playing Buzz #1')
pause(1);
set_amplitude(s,0);
pause(.5);
load_custom(s,yB,amp,pause_time);                       %
set_frequency(s,10) %this actually is 100Hz, but since the thing
repeats 10 times
disp('Now Playing Buzz #2')
pause(1);
set_amplitude(s,0);
pause(.5);
prompt = 'Were Buzz #1 and Buzz #2 the same? 9=SAME / 0=DIFFERENT
...then press ENTER ';
x = input(prompt);
     if x == 9;
         disp('Buzzes are marked as SAME')
     else
         disp('Buzzes are marked as DIFFERENT')
        SD = 1;
```

```
     end
pause(1);
clc;
%now record results
if f2==f3 || f2<f3 %if the stimuli are actually the same, or different
in the case f2<f3
    if SD == 0 %and the user marked them as the same
    evalc(sprintf('temp=SRM_%d_%d',TR(FqA,1),TR(FqB,1)));
    temp(1,2) = temp(1,2) + 1;
    evalc(sprintf('SRM_%d_%d=temp',TR(FqA,1),TR(FqB,1)));
    else %and the user marekd them as different
    evalc(sprintf('temp=SRM_%d_%d',TR(FqA,1),TR(FqB,1)));
    temp(1,1) = temp(1,1) + 1;
    evalc(sprintf('SRM_%d_%d=temp',TR(FqA,1),TR(FqB,1)));
    end
elseif f2>f3 %if stimuli are different but the first is greater, naming
convention wants them flipped
    if SD == 0 %and the user marked them as the same
    evalc(sprintf('temp=SRM_%d_%d;',TR(FqB,1),TR(FqA,1)));
    temp(1,2) = temp(1,2) + 1;
    evalc(sprintf('SRM_%d_%d=temp;',TR(FqB,1),TR(FqA,1)));
    else %and the user marekd them as different
    evalc(sprintf('temp=SRM_%d_%d;',TR(FqB,1),TR(FqA,1)));
    temp(1,1) = temp(1,1) + 1;
    evalc(sprintf('SRM_%d_%d=temp;',TR(FqB,1),TR(FqA,1)));
    end
end
end
Actual_Trials = [Trial_Index_A, Trial_Index_B];
for i = 1:numel(Actual_Trials)
Actual_Trials(i) = Threshold_results(Actual_Trials(i),1);
end
clc;
disp('All Trials Complete!')
clear F2 F3 FqA FqB Nf Nt Q SD SScompare TR Thrmax amp common f1 f2 f3
...
    i normvalue pause_time prompt q signal_length t temp x y y2 y3 yA
yB ...
    Trial_Index_A Trial_Index_B ans
shut_down;
```

Appendix D

pooling_psychophysics_results.m

```
%%Pools psychophysics results from individual
same_different_experiment.m runs.
%NDT 2017
% First load one complete regiment of psychophysical results into
workspace.
% All other results files (most likely representing other individuals)
that aren't in workplace are then to be added to the results that are
already in the workspace.
%user must manually update the name of the file they want to add.
filename = 'SD_results_16';  %file to add to workplace variables
%requires user to update manually filename to add
m = matfile(filename,'Writable',true);
varlist = who(m);
workspace_list = evalin('base','who'); %all the string names in
workspace
S = load(filename);
for n = 1:size(workspace_list,1)
    for j = 1:(size(varlist,1))
        if strcmp(workspace_list(n),varlist(j)) == 1
            evalc(sprintf( 'file = varlist(j)'));
            evalc(sprintf('toadd=S.%s',file{:}));
            evalc(sprintf('%s = %s + toadd',file{:},file{:}))
        end
    end
end
```

dPrime_from_HF.m

```
%%Gives the H and F values (Hit Rate and False-Alarm Rate) for data in
%workspace
%NDT 2017
%OUTPUTS Matrix of 4 columns
%First column corresponds to the frequency
%Second column corresponds to H value
%Third column is the F value
%Forth column is the total number of trials, N, that were performed
%Fifth column is the dprime value (same-difference, Independent-
Observation
%rule)
%ensure that the desired files are in the workspace
Frequency_Components = [110, 140, 150, 160, 190]; %these are the
frequencies you want to find H and F for
%use what ever naming convention is desired, for instance
%SRMs for simple
%SRMc for complex
F = Frequency_Components;
inputs = size(F,2);
H_F = [];
for f=1:inputs
```

```
   for q = 1:inputs
      if F(f) < F(q)
      temp=zeros(1,4);
      eval(sprintf('D1 = SRMc_%d_%d;',F(f),F(q)))  %ensure naming
convention is correct
      eval(sprintf('S1 = SRMc_%d_%d;',F(f),F(f)))  %ensure naming
convention is correct
      eval(sprintf('S2 = SRMc_%d_%d;',F(q),F(q)))  %ensure naming
convention is correct
      eval(sprintf('temp(1,1) = %d%d;',F(f),F(q)))
      temp(1,2) = D1(1,1)/(D1(1,1)+D1(1,2));%definition of H
      temp(1,3) = (S1(1,1)+S2(1,1))/(S1(1,1)+S2(1,1) +
S1(1,2)+S2(1,2));%definition of F
      temp(1,4) = S1(1,1)+S2(1,1) + S1(1,2)+S2(1,2) + D1(1,1)+D1(1,2);
      H_F = [H_F;temp];
      end
  end
end
d = zeros(size(H_F,1),1);
H_F = [H_F,d];
%% calculate dprim here
for c = 1:size(H_F,1)
    H = H_F(c,2);
    F = H_F(c,3);
    if H == 1 && F == 0
    H =.99;   %otherwise it'll come out to infinity
    end
    %Eq 4.4
    pc = (1/2)*H + (1/2)*(1-F);  %values SHOULD BE 1/2 and 1/2 !!!
    %Eq 9.4
    %pc = (pc^2) + ((1-pc)^2); << this makes things worse
    %Eq 9.7
    %pc = normcdf((norminv(H)-norminv(F))/2); %Probability of correct
Eq
    if pc < .5
    H_F(c,5) = 0;
    else
    %Eq 9.3
    dprime = 2*norminv(.5*(1+((2*pc)-1)^.5));
    H_F(c,5) = dprime;
    end
end
%% Clean up
Dprime_results = H_F;
disp('Frequency Comparison | H | F | N | dprime')
disp(Dprime_results)
disp('Frequency Comparison | H | F | N | dprime')
disp('note: open workspace variable to see true values of those listed
as zero here ')
clear c d D1 dprime f F Frequency Components H H_F inputs pc q S1 S2
temp
```

Appendix E

sum_strain_tensors.m

```
%%Takes time-series strain tensor matrices and adds them to make chords
%it also interpolates and formats for "sigmoid_Processing_Noise_Sweep.m
%Therefore, it should also be used to make "simple waveforms"... see
line with comment "SIMPLE WAVEFORMS"
%JCQ March 1 2016
%Modified by NDT April 3 2017
% >>takes comsol outputs of e11 e12 e13, etc (strain components) for
each
% frequency and re-samples (interpolate and extrapolate) to 1 second at
% 40000 samples
% >> then adds them to the first (base) frequency
% >> then  calculates first principal strain and picks out MAXIMUM POS
STRAIN AT EACH TIME PT
% >> makes graphs and saves data (may have to change save location)
clear all; close all; clc;
freq_now = [100,110,140,150,160,190]; %first one is the "base"
frequency that others will be added to
%% load strain tensor for first (base) frequency
    dir_name=sprintf('strain_tensors/%dhz',freq_now(1));
    e11_name=sprintf('%s/e11_reorg.txt',dir_name);
    e12_name=sprintf('%s/e12_reorg.txt',dir_name);
    e13_name=sprintf('%s/e13_reorg.txt',dir_name);
    e22_name=sprintf('%s/e22_reorg.txt',dir_name);
    e23_name=sprintf('%s/e23_reorg.txt',dir_name);
    e33_name=sprintf('%s/e33_reorg.txt',dir_name);
    e11_base=load(e11_name); clear e11_name;
    e12_base=load(e12_name); clear e12_name;
    e13_base=load(e13_name); clear e13_name;
    e22_base=load(e22_name); clear e22_name;
    e23_base=load(e23_name); clear e23_name;
    e33_base=load(e33_name); clear e33_name;
    E_base = {e11_base,e12_base,e13_base,e22_base,e23_base,e33_base};
%create a big structure
%% resample base (and carry it out for 1 second at 40000 samples per
second)
    for e = 1:6
    freq = freq_now(1);
    file = E_base{e}; %do this process to each matrix of struture
    theta=file(:,1); %timestep (phase angle)
    dend0=file(:,2); %mid, +y
    dend1=file(:,3); %mid, -y
    dend2=file(:,4); %distal, top
    dend3=file(:,5); %distal, +y
    dend4=file(:,6); %distal, -y
    theta_time=linspace(0,1/freq,length(theta));
    sig_time=linspace(0,1/freq,length(file)); %in seconds
    dt=0.025; %enter desired dt (in ms) **SAME AS NEURON
    tstop=1000; %enter desired tstop (in ms) **SAME AS NEURON
    dt=dt/1000; tstop=tstop/1000; %convert to seconds - don't change
    time=[0:dt:1/freq-dt]'; %in seconds);
```

```
    %Interpolate to get values at dt times!
    new_data=zeros(length(time),6);
    new_data(:,1)= interp1(theta_time,file(:,1),time);
    new_data(:,2)=interp1(theta_time,file(:,2),time);
    new_data(:,3)=interp1(theta_time,file(:,3),time);
    new_data(:,4)=interp1(theta_time,file(:,4),time);
    new_data(:,5)=interp1(theta_time,file(:,5),time);
    new_data(:,6)=interp1(theta_time,file(:,6),time);
    %Now round how many times you need to repeat this:
    rep_num=round(tstop/(1/freq))+1;
        all_data=repmat(new_data,rep_num,1);
        all_time=[0:dt:tstop];
        if length(all_data)<length(all_time);
            clear all_data
            all_data=repmat(new_data,rep_num*10,1);
        end
        all_data((length(all_time)+1):end,:)=[];
    E_base{e} = []; %this clears out the entire thing so it can be
saved over, since update is smaller than the original
    E_base{e} = all_data;  %the update is smaller than the original
because we are only using 5 filopodia, not 10)
    end
%% now go through the others (non-base) frequencies
for f = 2:size(freq_now,2)
disp(freq_now(f))
    %Load strain tensor data
    dir_name=sprintf('strain_tensors/%dhz',freq_now(f));
    e11_name=sprintf('%s/e11_reorg.txt',dir_name);
    e12_name=sprintf('%s/e12_reorg.txt',dir_name);
    e13_name=sprintf('%s/e13_reorg.txt',dir_name);
    e22_name=sprintf('%s/e22_reorg.txt',dir_name);
    e23_name=sprintf('%s/e23_reorg.txt',dir_name);
    e33_name=sprintf('%s/e33_reorg.txt',dir_name);
    e11=load(e11_name); clear e11_name;
    e12=load(e12_name); clear e12_name;
    e13=load(e13_name); clear e13_name;
    e22=load(e22_name); clear e22_name;
    e23=load(e23_name); clear e23_name;
    e33=load(e33_name); clear e33_name;
    E = {e11,e12,e13,e22,e23,e33};
%% resample base (and carry it out for 1 second at 40000 samples per
second)

     for e = 1:6
    freq = freq_now(f);
    file = E{e}; %do this process to each matrix of struture
    theta=file(:,1); %timestep (phase angle)
    dend0=file(:,2); %mid, +y
    dend1=file(:,3); %mid, -y
    dend2=file(:,4); %distal, top
    dend3=file(:,5); %distal, +y
    dend4=file(:,6); %distal, -y
    theta_time=linspace(0,1/freq,length(theta));
    sig_time=linspace(0,1/freq,length(file)); %in seconds
```

```
    dt=0.025; %enter desired dt (in ms) **SAME AS NEURON
    tstop=1000; %enter desired tstop (in ms) **SAME AS NEURON
    dt=dt/1000; tstop=tstop/1000; %convert to seconds - don't change
    time=[0:dt:1/freq-dt]'; %in seconds);
    %Interpolate to get values at dt times!
    new_data=zeros(length(time),6);
    new_data(:,1)= interp1(theta_time,file(:,1),time);
    new_data(:,2)=interp1(theta_time,file(:,2),time);
    new_data(:,3)=interp1(theta_time,file(:,3),time);
    new_data(:,4)=interp1(theta_time,file(:,4),time);
    new_data(:,5)=interp1(theta_time,file(:,5),time);
    new_data(:,6)=interp1(theta_time,file(:,6),time);
    %Now round how many times you need to repeat this:
    rep_num=round(tstop/(1/freq))+1;
        all_data=repmat(new_data,rep_num,1);
        all_time=[0:dt:tstop];
        if length(all_data)<length(all_time);
            clear all_data
            all_data=repmat(new_data,rep_num*10,1);
        end
        all_data((length(all_time)+1):end,:)=[];
    E{e} = [];
    E{e} = all_data;
    end
%% ADD TO BASE...
for c = 1:6
    %E{c} = E{c} + E_base{c}; %comment out this line for SIMPLE
WAVEFORMS
    rec = E{c};
    rec(:,1) = []; %delete theta column
    E{c} = rec;
end
%...and get some labels straight for the next section
e11 = E{1};
e12 = E{2};
e13 = E{3};
e22 = E{4};
e23 = E{5};
e33 = E{6};
%% Calculate principal strain
ns = length(e11(:,1)); %ns refers to the number of samples
num_points=size(e11,2);
dir_strain=zeros(ns,num_points); %to store the max pos strain in
for n_time=1:ns
        clear strain_matrix
        strain_matrix=zeros(6,num_points);
        strain_matrix(1,:)=e11(n_time,1:num_points);
        strain_matrix(2,:)=e12(n_time,1:num_points);
        strain_matrix(3,:)=e13(n_time,1:num_points);
        strain_matrix(4,:)=e22(n_time,1:num_points);
        strain_matrix(5,:)=e23(n_time,1:num_points);
        strain_matrix(6,:)=e33(n_time,1:num_points);
        %for each strain location, assemble vector and calc eigenvalues
        %NOTE: the below code is TERRIBLY INEFFICIENT!!
```

```
        for n_strain=1:num_points %looping through filopodia
            clear tensor eigenvalues max_val max_loc;
            tensor=[strain_matrix(1,n_strain) strain_matrix(2,n_strain)
strain_matrix(3,n_strain);...
                strain_matrix(2,n_strain) strain_matrix(4,n_strain)
strain_matrix(5,n_strain); ...
                strain_matrix(3,n_strain) strain_matrix(5,n_strain)
strain_matrix(6,n_strain)];
            %eigenvalues = eig(tensor);
            %[max_val,max_loc]=max(eigenvalues);
            if n_strain == 1; %then its dend 0
                dir = [0,1,0]; %positive y direction
                strain = dot(dir*tensor,dir);
            elseif n_strain == 2; %then its dend 1
                dir = [0,-1,0]; %negative y direction
                strain = dot(dir*tensor,dir);
            elseif n_strain == 3; %then its dend 2
                dir = [sqrt(2)/2,0,sqrt(2)/2]; %45 deg +z (x-z plane)
                strain = dot(dir*tensor,dir);
            elseif n_strain == 4; %then its dend 3
                dir = [0,sqrt(2)/2,-sqrt(2)/2]; %45 deg in +y (y-z
plane)
                strain = dot(dir*tensor,dir);
            elseif n_strain == 5; %then its dend 4
                dir = [0,-sqrt(2)/2,-sqrt(2)/2]; %45 deg in -y (y-z
plane)
                strain = dot(dir*tensor,dir);
            end
            dir_strain(n_time,n_strain)=strain;
        end
end
%% Zero the principal strain where the waveform would be negative
dir_strain(dir_strain<0) = 0;
%% Plot principal strain results and SAVE
figure;
 plot(dir_strain(:,1),'r'); hold on;
 plot(dir_strain(:,2),'b'); hold on;
 plot(dir_strain(:,3),'k'); hold on;
 plot(dir_strain(:,4),'c'); hold on;
 plot(dir_strain(:,5),'m'); hold on;
xlabel('samples'); ylabel('strain');
legend('Dend0','Dend1','Dend2','Dend3','Dend4');%'ICtop','neurite1','ne
urite2','bulbtop','bulbmid');
 title_name=sprintf('Directional strain calculated for %d+%d Hz
Simulations',freq_now(1),freq_now(f));
 title(title_name);
save_name=sprintf('PC_world/SimPsychophysics/zeroed_dir_strain_%dhz.txt
',freq_now(f));
%save_name=sprintf('PC_world/SimPsychophysics/zeroed_dir_strain_%d%dhz.
txt',freq_now(1),freq_now(f));
save(save_name,'dir_strain','-ascii');
clear save_matrix save_name;
end
clear all;
```

Appendix F

Sigmoid_Processing_Noise_Sweep.m

```
%%Converts filopodia strains into current injections
%MODIFIED from JCQ Feb 23, 2016, by NDT (2017) to be used after
sum_strain_tensors.m
%Reads in output .txt file from Comsol that shows zeroed-out principal
strain at all of the locations
%can run <reshape_strain.m> before this if working straight from comsol
    %- zeroes out the data
%make vectors to  input into Neuron
%next run <make_neuron_files.m>
%****Change dt, tstop (make these two same as neuron)
clear all; close all; clc;
%% NEED TO CHANGE:
% 1- folders that are "cd"-ed at beginning `
% 2- path added that contains the sigmoid.m function
% 3- directory name (saving output)
% 4- don't need to change any info in sigmoid section unless run COMSOL
-
% 5- input frequencies and amplitudes below
% 6- change dt and tstop if changing these number in NEURON... if not
don't
% change
%% Input frequencies, amplitudes, noise levels, and reps below:
%Note: It is suggested that reps be even numbered
%Note: SNRs are a function of noise level, amp level, freq, and chord
frequencies = [110,140,150,160,190,100110,100140,100150,100160,100190];
%THESE NUMBERS DON'T HAVE TO BE ACTUAL FREQUENCIES, but they must
correspond to principal strains
amplitudes = [50,25]; %each new entry will be applied to all
frequencies
%NOTE: the "amplitudes.txt" file that this script creates will be 2X as
long as needed.  So...
%...AFTER RUNNING, OPEN THE amplitudes.txt FILE AND MANUALLY DELETE
HALF THE ENTRIES (THE SECOND HALF)!!
noiselevels = [.0,05,1,3,5,10,15,20,30,50,75,100]; %each new entry will
be applied to all frequencies
rep = 10; %how many replicates should be performed
%% ENTER WHERE YOU WANT TO SAVE THIS DATA TO
dir_name =
sprintf('/Users/BarocasLab/Documents/MATLAB/PCpsychophys/SimPsychophysi
cs/Simulations/');
mkdir(dir_name);
%%
NL = noiselevels;
a = size(NL,2)*size(amplitudes,2)*rep;
b = size(frequencies,2);
SNRs = zeros(a,b);
clear a b
for n_freq=1:length(frequencies);
    freq=frequencies(n_freq);
    close all;
```

```
    cd
/Users/BarocasLab/Documents/MATLAB/PCpsychophys/SimPsychophysics/Simula
tions/PrincipalStrains %enter folder location of strain files
file_name=sprintf('zeroed_princ_strain_%dhz.txt',freq); %enter naming
convention for strain files
file=load(file_name); clear file_name;
%%
%Based off of points_list - don't change unless changed in COMSOL code
dend0=file(:,1); %mid, +y
dend1=file(:,2); %mid, -y
dend2=file(:,3); %distal, top
dend3=file(:,4); %distal, +y
dend4=file(:,5); %distal, -y
clear file;
addpath /Users/BarocasLab/Desktop/Codes_for_Info_Theory/Neuron_code
%Don't change below
Hao_max=8.5; %max on x axis, was 9, changed to 8.5 on 3/17/16
x = [0:.01:Hao_max]';
activ_stimulus_midpoint=5.6; %from Hao paper
steepness_factor = 0.81; %from Hao paper
y = sigmoid(x,1/steepness_factor,activ_stimulus_midpoint);
%figure; plot(x,y); title('Sigmoid for Rapid Channel from Hao and
Delmas')
x_new=linspace(0,0.2598,length(y))';%Change number - max value of
Comsol strain outputs!!! only change if something changes in COMSOL
new_midpoint=(activ_stimulus_midpoint/Hao_max)*max(x_new);
clear fit_coeff fit_eq a
fit_eq=fittype('1./(1+exp(-a*(x_new-
.1712)))','dependent',{'y'},'independent',{'x_new'},'coefficients',{'a'
}); %change midpoint
[myfit,goodness]=fit(x_new,y,fit_eq) %use "a" below!!!!!
%figure;plot(myfit,x_new,y); title('Fit for Hao & Delmas data');
clear fit_coeff fit_eq  goodness x
sigmoid_x = x_new; clear x_new y;
%CHANGE VALUES BELOW IF CHANGED SOMETHING ABOVE (IN X_NEW)!!!!
%transition_location = 0.1712; clear new_midpoint
activ_stimulus_midpoint; %from 'new_midpoint' above
%transition_degree = 40.39; clear steepness_factor; %from 'myfit'
output "a"
transition_degree = myfit.a;
transition_location = new_midpoint;
clear myfit a steepness_factor new_midpoint activ_stimulus_midpoint;
%just to initiate matrix
%LOOP THROUGH AMPLITUDES (Change amp then pass through sigmoid!)
%for n_amp = 1:length(amplitudes)  %IF YOU COMMENT THIS LINE BACK IN,
MAKE
%SURE TO ALSO GET THE END STATEMENT, AND amp = amplitudes(n_amp)
if n_freq <= length(frequencies)/2
amp = amplitudes(1);
else
amp = amplitudes(2);
end
n_amp=1;
%THE ABOVE WAS TO SIMPLY AMPLITUDE
```

```
    for nz = 1:length(NL)
        for r = 1:rep
    SignalNoiseRatios = [];
    %amp = amplitudes(n_amp); clear dend0_scaled dend1_scaled
dend2_scaled dend3_scaled dend4_scaled;
    dend0_scaled=dend0.*amp;
    dend1_scaled=dend1.*amp;
    dend2_scaled=dend2.*amp;
    dend3_scaled=dend3.*amp;
    dend4_scaled=dend4.*amp;
    %adding noise
    %first generate matrix to put noise
    Vert = size(dend1,1);
    dend0_noise = normrnd(0,NL(nz),[Vert,1]);
    dend1_noise = normrnd(0,NL(nz),[Vert,1]);
    dend2_noise = normrnd(0,NL(nz),[Vert,1]);
    dend3_noise = normrnd(0,NL(nz),[Vert,1]);
    dend4_noise = normrnd(0,NL(nz),[Vert,1]);
    %for Nz = 1:size(dend0,1) %adding noise after amplifing signal
    %dend0_noise(Nz) = dend0_noise(Nz)*normrnd(0,NL(n_amp)); %indexing
noise against amplitude
    %dend1_noise(Nz) = dend1_noise(Nz)*normrnd(0,NL(n_amp));
    %dend2_noise(Nz) = dend2_noise(Nz)*normrnd(0,NL(n_amp));
    %dend3_noise(Nz) = dend3_noise(Nz)*normrnd(0,NL(n_amp));
    %dend4_noise(Nz) = dend4_noise(Nz)*normrnd(0,NL(n_amp));
    %end
   %then calculate SNR
   SNR =
mean([snr(dend0_scaled,dend0_noise),snr(dend1_scaled,dend1_noise),snr(d
end2_scaled,dend2_noise),snr(dend3_scaled,dend3_noise),snr(dend4_scaled
,dend4_noise)]);
   %then add the noise
    dend0_scaled=dend0_scaled+dend0_noise;
    dend1_scaled=dend1_scaled+dend1_noise;
    dend2_scaled=dend2_scaled+dend2_noise;
    dend3_scaled=dend3_scaled+dend3_noise;
    dend4_scaled=dend4_scaled+dend4_noise;
dend0_out = sigmoid(dend0_scaled,transition_degree,
transition_location);
dend1_out = sigmoid(dend1_scaled,transition_degree,
transition_location);
dend2_out = sigmoid(dend2_scaled,transition_degree,
transition_location);
dend3_out = sigmoid(dend3_scaled,transition_degree,
transition_location);
dend4_out = sigmoid(dend4_scaled,transition_degree,
transition_location);
clear dend0_scaled dend1_scaled dend2_scaled dend3_scaled dend4_scaled
clear dend0_noise dend1_noise dend2_noise dend3_noise dend4_noise
% %UNCOMMENT BELOW TO PLOT OUTPUTS
% figure;
% plot(theta,dend0_scaled,'r'); hold on; plot(theta,dend0_out,'b');
legend('Strains','Sigmoid Output'); title('Dendrite
0');xlabel('theta'); ylabel('strains');
```

```
% figure;
% plot(theta,dend1_scaled,'r'); hold on; plot(theta,dend1_out,'b');
legend('Strains','Sigmoid Output'); title('Dendrite
1');xlabel('theta'); ylabel('strains');
% figure;
% plot(theta,dend2_scaled,'r'); hold on; plot(theta,dend2_out,'b');
legend('Strains','Sigmoid Output'); title('Dendrite
2');xlabel('theta'); ylabel('strains');
% figure;
% plot(theta,dend3_scaled,'r'); hold on; plot(theta,dend3_out,'b');
legend('Strains','Sigmoid Output'); title('Dendrite
3');xlabel('theta'); ylabel('strains');
% figure;
% plot(theta,dend4_scaled,'r'); hold on; plot(theta,dend4_out,'b');
legend('Strains','Sigmoid Output'); title('Dendrite
4');xlabel('theta'); ylabel('strains');
data=horzcat(dend0_out,dend1_out,dend2_out,dend3_out,dend4_out);
code = (size(NL,2)*rep*(n_amp-1)) + (rep*(nz-1)) + r; %I call this
crazy indexing
SNRs(code,n_freq) = SNR;
%Now save the data
file_name= sprintf('%dhz_dend0_%d.txt',freq,code);
full_file_name=fullfile(dir_name,file_name);
fid = fopen(full_file_name,'w');
fprintf(fid,'%d\n',data(:,1));
fclose(fid);
file_name= sprintf('%dhz_dend1_%d.txt',freq,code);
full_file_name=fullfile(dir_name,file_name);
fid = fopen(full_file_name,'w');
fprintf(fid,'%d\n',data(:,2));
fclose(fid);
file_name= sprintf('%dhz_dend2_%d.txt',freq,code);
full_file_name=fullfile(dir_name,file_name);
fid = fopen(full_file_name,'w');
fprintf(fid,'%d\n',data(:,3));
fclose(fid);
file_name= sprintf('%dhz_dend3_%d.txt',freq,code);
full_file_name=fullfile(dir_name,file_name);
fid = fopen(full_file_name,'w');
fprintf(fid,'%d\n',data(:,4));
fclose(fid);
file_name= sprintf('%dhz_dend4_%d.txt',freq,code);
full_file_name=fullfile(dir_name,file_name);
fid = fopen(full_file_name,'w');
fprintf(fid,'%d\n',data(:,5));
fclose(fid);
%clearvars -except n_freq freq frequencies n_amplitudes n_amp
amplitudes dir_name SNRs dend0 dend1 dend2 dend3 dend4 NL;
         end %rep loop
    end %noise loop (NL)
%end %amplitude loop %changed to an if state
end %frequency loop
%amp_list = amplitudes';
%extending this to account for noise
```

```
noise_number = size(NL,2);
amplitude_number = size(amplitudes,2);
REPEAT = noise_number*rep*amplitude_number;
amp_list = linspace(1,REPEAT,REPEAT)';
freq_list = frequencies';
file_name = sprintf('frequencies_list.txt');
full_file_name = fullfile(dir_name,file_name);
fid = fopen(full_file_name,'w');
fprintf(fid,'%d\n',freq_list);
fclose(fid);
file_name = sprintf('amplitude_list.txt');
full_file_name = fullfile(dir_name,file_name);
fid = fopen(full_file_name,'w');
fprintf(fid,'%d\n',amp_list);
fclose(fid);
%save(SNRs);
%SNRs_list = SNRs';
%file_name = sprintf('SNRs_list.txt');
%full_file_name = fullfile(dir_name,file_name);
%fid = fopen(full_file_name,'w');
%fprintf(fid,'%d\n',SNRs_list);
%fclose(fid);
```

sigmoid.m

```
%JCQ 02-24-16
%x is function you want to pass through sigmoid
%a is a degree to which the curve goes from 0 to 1 (larger a = sharper)
%b is where the curve happens (on the x axis)
function y = sigmoid(x,a,b)
y = 1 ./ (1 + exp(-a*(x-b)));
end
```

Collate.m

```
function [ Store ] = Collate( New,Store )
%This function takes a "New" row vector of any size and appends it as
the last line of a matrix called "Store"  of any size (including an
empty "store" matrix, i.e size of 0).
%%%NOTE:THIS FUNCTION USES ZERO (0) AS A PLACE HOLDER i.e. it
"zeropads" matrices !!
    if (size(New,2) < size(Store,2))
        S = size(Store,2) - size(New,2);
        zeropad = zeros(size(New,1),S);
        New = [New zeropad];
    elseif size(Store,2) < size(New,2) && size(Store,2) > 0
        S = size(New,2) - size(Store,2);
        zeropad = zeros(size(Store,1),S); %again, zeros are used as
place holder
        Store = [Store zeropad];
    end
    Store = [Store;New];
End
```

Appendix G

make_neuron_files.m

```
%JCQ Sept 24 2015
%Code used to set-up NEURON program to run at different parameters
%load a NEURON hoc file, make a new folder and place copies of the hoc
file
%(with a parameter changed) into the new folder
%CHANGE:
% 1- neuron_Currents_folder = where the currents are stored (from
% extrapolate_sigmoid.m)
% 2- directory_name_prefix and directory_name if want to change folders
clc; clear all; close all;
%% Input location of currents folder - CHANGE
neuron_currents_folder=sprintf('/Users/BarocasLab/Documents/MATLAB/PCps
ychophys/SimPsychophysics/Simulations/'); %CHANGE
%%
freq_vector_name = sprintf('frequencies_list.txt');
amp_vector_name = sprintf('amplitude_list.txt');
frequencies_vector =
load(fullfile(neuron_currents_folder,freq_vector_name));
amplitudes = load(fullfile(neuron_currents_folder,amp_vector_name));
output_vector=amplitudes; clear amplitudes freq_vector_name
amp_vector_name;
for n_freq_vec=1:length(frequencies_vector)
    clearvars -except frequencies_vector n_freq_vec output_vector
neuron_currents_folder;
freq_to_write = num2str(frequencies_vector(n_freq_vec)); %CHANGE
freq_num=str2num(freq_to_write);
directory_name_prefix=['neuron_code_']; %CHANGE
directory_name=[directory_name_prefix,freq_to_write,'hz']; %CHANGE
mkdir(directory_name);
results_directory=['results'];
results_fullname=[directory_name,'/',results_directory];
mkdir(results_fullname);
input_file_name=['160317_blank.hoc']; %CHANGE IF CHANGED NEURON_CODE
n_amplitudes=length(output_vector); %CHANGE
for n_output=1:n_amplitudes;
    amp_to_write=num2str(n_output);
output_file_name_prefix=[directory_name, '/runcode_']; %CHANGE
results_file_name_prefix = [results_directory, '/'
,directory_name_prefix,'results_'];
current_file_name_prefix = [results_directory, '/'
,directory_name_prefix,'current_'];
ap_file_name_prefix = [results_directory, '/'
,directory_name_prefix,'AP_'];
%Move currents
%file_now=sprintf('/Users/julia/Dropbox/Neuron_dropbox/%s/%dhz_dend0_%d
.txt',neuron_currents_folder,freq_num,n_output);
file_now = fullfile(neuron_currents_folder,
sprintf('%dhz_dend0_%d.txt',freq_num,n_output));
copyfile(file_now,directory_name);clear file_now;
```

```
file_now = fullfile(neuron_currents_folder,
sprintf('%dhz_dend1_%d.txt',freq_num,n_output));
copyfile(file_now,directory_name);clear file_now;
file_now = fullfile(neuron_currents_folder,
sprintf('%dhz_dend2_%d.txt',freq_num,n_output));
copyfile(file_now,directory_name);clear file_now;
file_now = fullfile(neuron_currents_folder,
sprintf('%dhz_dend3_%d.txt',freq_num,n_output));
copyfile(file_now,directory_name);clear file_now;
file_now = fullfile(neuron_currents_folder,
sprintf('%dhz_dend4_%d.txt',freq_num,n_output));
copyfile(file_now,directory_name);clear file_now;
    fidr = fopen(input_file_name,'r');
    output_file_name=[output_file_name_prefix,
num2str(n_output),'.hoc'];
    fidw = fopen(output_file_name,'w');
    nextLine = fgets(fidr);  % Get the first line of input
    %amp_to_write=num2str(output_vector(n_output));
    file_to_write=[results_file_name_prefix,num2str(n_output),'.dat'];

file_to_write_current=[current_file_name_prefix,num2str(n_output),'.dat
'];
    ap_file_to_write = [ap_file_name_prefix,num2str(n_output),'.dat'];
    while nextLine >= 0  % Loop until getting -1 (end of file)
        nextLine = strrep(nextLine,'AMP_TO_CHANGE',amp_to_write);
        nextLine = strrep(nextLine,'FREQ_TO_CHANGE',freq_to_write);
        %nextLine =
strrep(nextLine,'THRESH_TO_CHANGE',thresh_to_write);
        nextLine = strrep(nextLine, 'APFILE_NAME', ap_file_to_write);
        nextLine = strrep(nextLine, 'results/sinevec.dat',
file_to_write_current);
        nextLine = strrep(nextLine, 'results/091815_results_1.dat',
file_to_write);
        fprintf(fidw,'%s',nextLine); %Write the line to the output file
        nextLine = fgets(fidr);  %Get the next line of input
    end
    fclose(fidw);
    fclose(fidr);
end
%write amplitudes to a txt file
amplitude_file_name=[directory_name,'/amplitude_vector.txt'];
save(amplitude_file_name,'output_vector','-ascii');
%%
%Make the run file... type in chmod 775 run_neuron_code.sh in command
then
%./run_neuron_code.sh
words_matrix=[];
for n_print=1:n_amplitudes
    words=sprintf('nrngui runcode_%d.hoc -Py_NoSiteFlag',n_print);
    words_matrix=strvcat(words_matrix,words);
end
words_matrix;
shell_code_name=[directory_name,'/run_neuron_code.sh'];
```

```
export_path=sprintf('export PATH=/Applications/NEURON-
7.4/nrn/x86_64/bin:$PATH');  %CHANGE FOR NEURON VERSION
fileID=fopen(shell_code_name,'w');
fprintf(fileID,'%s\n',export_path(:));
for n_print=1:n_amplitudes
fprintf(fileID,'%s\n',words_matrix(n_print,:));
end
fclose(fileID);
end
```

160317_blank.hoc

```
//Building model from 042415_1.hoc and notes on 3D morphology
//JCQ 2015-2016
//////////////////////
/* model specification */
//////////////////////
/////// topology ////////
create axon, soma[3], dend[5]
////// connections /////////
connect axon(1), soma[0](0)
connect soma[1](0), soma[0](1) //connect left end of 1 to right end of
0
connect soma[2](0), soma[1](1) //connect left end of 2 to right end of
1
connect dend[0](0), soma[0](1) // 0 is left
connect dend[1](0), soma[0](1) // 1 is right
connect dend[2](0), soma[2](1) // 2 is middle of the 3
connect dend[3](0), soma[2](1) // 3 is on the left of the 3
connect dend[4](0), soma[2](1) // 4 is on the right of the 3
////// geometries /////////
axon {
      length_axon = 247
      diam = 5.8
      nseg = 4
      pt3dclear()
      for i = 0,nseg {
            pt3dadd(i*-length_axon/nseg, 0, 0, diam)
            }
      }
// first soma (on left side, connected to axon)
soma[0] {
      length_soma0=300
      soma_diam=3
      nseg=4
      pt3dclear()
      for i=0,nseg {
            pt3dadd(i*length_soma0/nseg, 0, 0, soma_diam)
            }
      }
//second soma (on right side, connected to the bulbous ending)
soma[1] {
      length_soma1=300
      soma_diam = 3
      nseg=4
```

```
      for i=0,nseg {
            pt3dadd(length_soma0+i*length_soma1/nseg, 0, 0, soma_diam)
            }
      }
//ultra-terminal bulbous ending (on right side, connected to the 3
distal dendrites)
soma[2] {
      length_soma2=10
      diam = 10
      nseg=1
      for i=0,nseg {
            pt3dadd(length_soma0+length_soma1+i*length_soma2/nseg, 0,
0, diam)
            }
      }
dend[0] {
      length_dend0=1.6
      diam = .6
      nseg=1
      for i=0,nseg {
            //pt3dadd(length_soma0, i*length_dend0/nseg, 0, diam)
            pt3dadd(length_soma0, i*length_dend0/nseg*2, 0, diam)
//scaled because it didn't look long enough
                  //pt3dadd(length_soma0, soma_diam/2, 0, diam)
      //pt3dadd(length_soma0, soma_diam/2+length_dend0, 0, diam)
            }
      }
      dend[1] {
      length_dend1=1.6
      diam = .6
      nseg=1
      for i=0,nseg {
            //pt3dadd(length_soma0, -i*length_dend1/nseg, 0, diam)
            pt3dadd(length_soma0, -i*length_dend1/nseg*2, 0, diam)
//scaled because it didn't look long enough
            }
      }
dend[2] {
      length_dend2=2*1.6 //scaled because it didnt' look long enough
      scale_dend2=length_dend2/2 //because length of tet is 2
      //x_val=length_dend2*sqrt(3)/3
      diam = .6
      nseg=1
            pt3dadd(length_soma0+length_soma1+length_soma2, 0, 0,diam)
      pt3dadd((length_soma0+length_soma1+length_soma2)+(2*sqrt(2)/sqrt(
3))*scale_dend2, 0*scale_dend2, (sqrt(3)-1/sqrt(3))*scale_dend2, diam)

      }
      dend[3] {
      length_dend3=2*1.6  //scaled because it didn't look long enough
      scale_dend3=length_dend3/2 //because length of tet is 2
      diam = .6
      nseg=1
            pt3dadd(length_soma0+length_soma1+length_soma2, 0, 0,diam)
```

```
      pt3dadd((length_soma0+length_soma1+length_soma2)+(2*sqrt(2)/sqrt(
3))*scale_dend3, 1*scale_dend3, (-1/sqrt(3))*scale_dend3, diam)
      }
      dend[4] {
      length_dend4=2*1.6   //scaled because it didn't look long enough
      scale_dend4=length_dend4/2 //because length of tet is 2
      diam = .6
      nseg=1
            pt3dadd(length_soma0+length_soma1+length_soma2, 0, 0,diam)
      pt3dadd((length_soma0+length_soma1+length_soma2)+(2*sqrt(2)/sqrt(
3))*scale_dend4, -1*scale_dend4, (-1/sqrt(3))*scale_dend4, diam)
      }
///// visualize shape//////
objref shape_plot
shape_plot = new Shape()
shape_plot.show(0)
dend[2] shape_plot.color(2)
dend[3] shape_plot.color(3)
//dend shape_plot.color(5)
//axon shape_plot.color(2)
 //dend[2] shape_plot.color(3)
 // dend[3] shape_plot.color(3)
 //dend[4] shape_plot.color(3)
  //dend[0] shape_plot.color(3)
 //dend[1] shape_plot.color(3)
//////insert HH ///////
// DENDRITES
for i = 0, 4 dend[i] {
    insert hh
}
 /*
//SOMA
for i = 0, 1 soma[i] {
      insert hh
}
*/
//AXON
axon {
      insert hh
      gnabar_hh(0:1)=0.12:0.24 //double density of Na channels at axon
hillock
}
topology()
finitialize()
forall {
      print secname()
      for i = 0, n3d()-1 print i, x3d(i), y3d(i), z3d(i), diam3d(i)
      }
////PARAMETERS/////
dt=.025
tstop = 1000
v_init = -65
/////sinusoidal current//////
////// stimulate //////////
```

```
objectvar stim0, stim1, stim2, stim3, stim4
      //Dendrite 0
      dend[0] stim0 = new IClamp(.5) //can change where the stim is
applied
      stim0.del = 0
      stim0.dur = tstop
      stim0.amp = 100
      //Dendrite 1
      dend[1] stim1 = new IClamp(.5)
      stim1.del = 0
      stim1.dur = tstop
      stim1.amp = 100
      // Dendrite 2
      dend[2] stim2 = new IClamp(.5)
      stim2.del = 0
      stim2.dur = tstop
      stim2.amp = 100
            //Dendrite 3
      dend[3] stim3 = new IClamp(.5)
      stim3.del = 0
      stim3.dur = tstop
      stim3.amp = 100
      //Dendrite 4
      dend[4] stim4 = new IClamp(.5)
      stim4.del = 0
      stim4.dur = tstop
      stim4.amp = 100
////////////////////////////
/////*LOAD DEND CURRENT *////
////////////////////////////
size=tstop/dt
objref file_dend0, file_dend1, file_dend2, file_dend3, file_dend4
file_dend0 = new File()
file_dend1 = new File()
file_dend2 = new File()
file_dend3 = new File()
file_dend4 = new File()
file_dend0.ropen("FREQ_TO_CHANGEhz_dend0_AMP_TO_CHANGE.txt") //change
frequency
file_dend1.ropen("FREQ_TO_CHANGEhz_dend1_AMP_TO_CHANGE.txt") //change
frequency
file_dend2.ropen("FREQ_TO_CHANGEhz_dend2_AMP_TO_CHANGE.txt") //change
frequency
file_dend3.ropen("FREQ_TO_CHANGEhz_dend3_AMP_TO_CHANGE.txt") //change
frequency
file_dend4.ropen("FREQ_TO_CHANGEhz_dend4_AMP_TO_CHANGE.txt") //change
frequency
objectvar data_dend0
      data_dend0= new Vector(size)
      data_dend0.scanf(file_dend0)
objectvar data_dend1
      data_dend1= new Vector(size)
      data_dend1.scanf(file_dend1)
objectvar data_dend2
```

```
      data_dend2= new Vector(size)
      data_dend2.scanf(file_dend2)
objectvar data_dend3
      data_dend3= new Vector(size)
      data_dend3.scanf(file_dend3)
objectvar data_dend4
      data_dend4= new Vector(size)
      data_dend4.scanf(file_dend4)
objectvar data_dend0_scaled
      data_dend0_scaled = new Vector(size)
objectvar data_dend1_scaled
      data_dend1_scaled = new Vector(size)
objectvar data_dend2_scaled
      data_dend2_scaled = new Vector(size)
objectvar data_dend3_scaled
      data_dend3_scaled = new Vector(size)
objectvar data_dend4_scaled
      data_dend4_scaled = new Vector(size)
//////////////////////
////* SCALES AND     *///
////* ASSIGNS CURRENT*///
//////////////////////
data_dend0_scaled=data_dend0.mul(5) //can change scalar
data_dend1_scaled=data_dend1.mul(5)
data_dend2_scaled=data_dend2.mul(5)
data_dend3_scaled=data_dend3.mul(5)
data_dend4_scaled=data_dend4.mul(5)
      //makes the current equal the imported value
      data_dend0_scaled.play(&stim0.amp, dt)
      data_dend1_scaled.play(&stim1.amp, dt)
      data_dend2_scaled.play(&stim2.amp, dt)
      data_dend3_scaled.play(&stim3.amp, dt)
      data_dend4_scaled.play(&stim4.amp, dt)
//////////////////////
/* COUNT ACTION      */
/*   POTENTIALS      */
//////////////////////
objectvar apc
axon apc= new APCount(1)
objref APVec
APVec = new Vector ()
//apc.thresh = -10
apc.record(APVec)
objref file5 //save APs
file5 = new File()
file5.wopen("APFILE_NAME")
// graphical display
objref g
g = new Graph()
g.size(0,tstop, -80, 40)
/*g.addvar("dendrite 2", "dend[2].v(0.5)", 1, 1)*/ //first # color,
second # thickness
/*g.addvar("soma 0", "soma[0].v(0.5)", 2, 1) */
g.addvar("axon", "axon.v(1)", 3, 1)
```

```
////// save data/////
objref rec_vector, rec_time_vector
rec_vector = new Vector(size)
rec_time_vector = new Vector(size)
rec_vector.record(&axon.v(1))   //OUTPUT
rec_time_vector.record(&t)
objref file3 //save time vector
file3 = new File()
file3.wopen("results/091815_results_1.dat")
objref file4 //save time vector
file4 = new File()
file4.wopen("results/sinevec.dat")
//////////////////////
/* run controls      */
//////////////////////
proc initialize() {
      t = 0
      finitialize(v_init)
      fcurrent()
      }
proc integrate() {
      g.begin()
      while (t<tstop){
            fadvance()
            g.plot(t)
      }
      g.flush()
}
proc go() {
      initialize()
      integrate()
      //rec_vector.printf(file3)
      for i=0,size-1 {
      file3.printf("%g %g\n", rec_time_vector.x(i), rec_vector.x(i))
            file4.printf("%g %g %g %g %g %g\n",rec_time_vector.x(i),
data_dend0_scaled.x(i),data_dend1_scaled.x(i),data_dend2_scaled.x(i),da
ta_dend3_scaled.x(i),data_dend4_scaled.x(i))
      }
      file3.close()
      file4.close()
      file_dend0.close()
      file_dend1.close()
      file_dend2.close()
      file_dend3.close()
      file_dend4.close()
file5.printf("%g %g %g", apc.thresh, apc.time, apc.n)
      file5.close()
}
go()
quit()
```

Appendix H

Results_Processing_dprime.m

```
dprimesALL = [];
%NDT March 2017
%modified version of load_results_interval.m
for noiseOFinterest = 1:10 %...if 10 noise levels are used
directory_name_prefix='neuron_code_'; %CHANGE!!!
%DIRECTORY FREQUENCY IS WHAT SPECIFIES WHICH RESULTS TO ANALYZE:
%directory_freq = [100110,100140,100150,100160,100190]%CHANGE!!!!
%directory_freq = [110, 140, 150, 160, 190]%CHANGE!!!!
directory_freq = [110, 140, 150, 160, 190, 100110, 100140, 100150,
100160, 100190]%CHANGE!!!!
ss = 10; %sub second rate: 10 corresponds to FR100
AP_thresh = 0;
ampOFinterest = 1;
%these are for reading/indexing purposes, not writing/creating purposes
noiselvlNumber=12;  %this need to reflect the experimental set-up
ampsNumber = 1;   %this need to reflect the experimental set-up
reps =10;   %this need to reflect the experimental set-up
%% For initializing (don't need to change)
Rate = [];
SSRs = [];
ISIs = [];
voltage_store =[];
name_freq = directory_freq;
% if directory_freq(1) > 1000 %this indicates the naming convention
% name_freq = directory_freq - 100000;  %and corrects for it
% end
%% Load files
for f = 1:size(directory_freq,2) %added for purposes of comparing
different frequencies
directory_name = [directory_name_prefix, num2str(directory_freq(f)),
'hz']; %created in <make_neuron_files.m>
amplitude_name=[directory_name, '/amplitude_vector.txt']; %takes the
directory_name, appends specific .txt name
amplitude_vector=load(amplitude_name);  %THIS MAKES THE .txt into a a
MAT  <---- .txt conversion "load()"
number_files=length(amplitude_vector); %the number of different force
amplitudes
for n_output=1:number_files;
    input_file_name = [directory_name, '/results/'
directory_name_prefix, 'results_',num2str(n_output),'.dat'];
    %Load File
    data = load(input_file_name);
    time=data(:,1);
    voltage=data(:,2);
    voltage_store=[voltage_store voltage];
%% Extract info
    %Find APs
    threshold = AP_thresh; %was-13
```

```
    [peaks,loc]=findpeaks(voltage,'MinPeakDistance',70,
'MinPeakHeight', -13); %this min peak height is important, initially
was 5
    %loc_hold(1:size(loc),n_output) = loc;
    [found_AP]=find(peaks>threshold);
    loc_AP=loc(found_AP);
    %time_AP=time(loc_AP);
    %Calculate and Store Frequencies
    frequency = size(peaks,1);
    Rate = Collate(frequency,Rate);
    %Sub-Second Rates
     lp = loc_AP; %lp is "the loc portion"
    SSRtemp = zeros(1,ss);
    sample_portion = (size(voltage_store,1)/ss);
    for sub = 1:ss
       lp = loc_AP(loc_AP<=(sub*sample_portion));
       lp = lp(lp>(sub-1)*sample_portion);
       SSRtemp(1,sub) = size(lp,1)/ss;
    end
    SSRs = [SSRs;SSRtemp];
    %Caclulate and Store Inter-Spike Intervals (ISIs)
    ISItemp = [];
    for isi = 1:(size(loc_AP)-1)
        ISItemp = [ISItemp ((loc_AP(isi+1))-loc_AP(isi))];
    end
    if size(ISItemp,2) == 0
    ISItemp = [0];
    end
    %COMBINE ISItemp matrices into ISIs matrix so that each row in ISIs
is a different input
    [ISIs] = Collate(ISItemp,ISIs); %ENSURE that "COLLATE" is in
directory!
%
%     %plots
%     figure;
%     plot(time,voltage,'k',loc/40,peaks,'o');
%     axis([20 220 -80 50]);
%     xlabel('time(ms)')
%     ylabel('voltage(mV)')
%     title_name =['Spikes from ' num2str(amplitude_vector(n_output)) '
amplitude stim at ' num2str(directory_freq(f)) ' Hz'];
%     title(title_name)
%     plot(time,threshold*ones(1,length(time)),'-r');
%
end
end
%% find spike distributions and calculate some stats
FRQno = size(directory_freq,2);
FR_byFrequency = zeros(FRQno,(size(SSRs,2)*reps));
ISI_byFrequency = zeros(FRQno,(size(ISIs,2)*reps));
FR_mean_SD = zeros(FRQno,2);
ISI_mean_SD = zeros(FRQno,2);
 for freq1 = 1:size(directory_freq,2) %pick a frequency, freq1
 for freq2 = 1:size(directory_freq,2) %pick another frequency,freq2
```

```
if freq1 < freq2 %only compare different frequencies and only compare
one time!!
code1 = (noiselvlNumber*reps*ampsNumber*(freq1-1)) +
(noiselvlNumber*reps*(ampOFinterest-1)) + (reps*(noiseOFinterest-1)) +
1;
code2 = (noiselvlNumber*reps*ampsNumber*(freq2-1)) +
(noiselvlNumber*reps*(ampOFinterest-1)) + (reps*(noiseOFinterest-1)) +
1;
FR_dist1 = SSRs(code1:code1+reps-1,:);
FR_dist2 = SSRs(code2:code2+reps-1,:);
FR_dist1 = reshape(FR_dist1,[1,numel(FR_dist1)]);
FR_dist2 = reshape(FR_dist2,[1,numel(FR_dist2)]);
%store ISI distribution from all replicates
FR_byFrequency(freq1,:) = FR_dist1;
FR_byFrequency(freq2,:) = FR_dist2;
ISI_dist1 = ISIs(code1:code1+reps-1,:);
ISI_dist2 = ISIs(code2:code2+reps-1,:);
ISI_dist1 = reshape(ISI_dist1,[1,numel(ISI_dist1)]);
ISI_dist2 = reshape(ISI_dist2,[1,numel(ISI_dist2)]);
%store ISI distribution from all replicates
ISI_byFrequency(freq1,:) = ISI_dist1;
ISI_byFrequency(freq2,:) = ISI_dist2;
%removing zeros from ISIs
ISI_dist1 = ISI_dist1(ISI_dist1~=0);  %removing zeros that were added
by collate.m
ISI_dist2 = ISI_dist2(ISI_dist2~=0);    %because you can't have an ISI
of zero
%mean
FR_mean_SD(freq1,1) =mean(FR_dist1);
FR_mean_SD(freq2,1) =mean(FR_dist2);
ISI_mean_SD(freq1,1)=mean(ISI_dist1);
ISI_mean_SD(freq2,1)=mean(ISI_dist2);
%standard deviation
FR_mean_SD(freq1,2) =std(FR_dist1);
FR_mean_SD(freq2,2) =std(FR_dist2);
ISI_mean_SD(freq1,2)=std(ISI_dist1);
ISI_mean_SD(freq2,2)=std(ISI_dist2);
end
end
 end
%% Using the mean and standard of firing activity find d'
dprimes = [];
for f1 = 1:size(directory_freq,2)
for f2 = 1:size(directory_freq,2)
if directory_freq(f1) < directory_freq(f2)
    m1 = FR_mean_SD(f1,1);
    m2 = FR_mean_SD(f2,1);
    s1 = FR_mean_SD(f1,2);
    s2 = FR_mean_SD(f2,2);
    FRd = ((m1-m2)/((.5*((s1^2) + (s2^2)))^.5));

    m1 = ISI_mean_SD(f1,1);
    m2 = ISI_mean_SD(f2,1);
    s1 = ISI_mean_SD(f1,2);
```

```
    s2 = ISI_mean_SD(f2,2);
    ISId = ((m1-m2)/((.5*((s1^2) + (s2^2)))^.5));

    dprime_temp = [FRd,ISId];
    dprimes = [dprimes;dprime_temp];
end
end
end
dprimes = abs(dprimes);
dprimesALL = [dprimesALL,dprimes];
end
%% clean up workspace
clearvars -except dprimes dprimesALL voltage_store SSRs ISIs
ISI_mean_SD FR_mean_SD FR_byFrequency ISI_byFrequency
```